Web程序员成功之路

Struts2 Web



- 迅速提高读者Web开发能力，全面挖掘读者开发潜力
- 一线资深Web程序员经验力作，窗内网独家推荐自学教材
- 16个小时视频教学，简化学习过程
- 60个实战案例与理论知识综合讲解，提高应用能力
- 网站互动教学（www.itzcn.com），QQ群在线帮助读者解疑

杨少敏　樊双灵　编著

随书附赠 DVD-ROM

清华大学出版社

Web 程序员成功之路

Struts 2 Web 开发学习实录

杨少敏　樊双灵　编著

清华大学出版社
北　京

内 容 简 介

本书介绍了比较流行的技术 Struts 2，全书分为 4 篇，分别为：Struts 2 基础篇、Struts 2 知识篇、Struts 2 应用篇和实例篇。Struts 2 基础篇(第 1～2 章)讲解了 Struts 2 的基础配置。Struts 2 知识篇(第 3～10 章)讲解了 Struts 2 的各种知识，如：数据类型转换、国际化、异常处理、拦截器、数据校验、OGNL、标签库、文件上传下载和避免表单重复提交等。Struts 2 应用篇(第 11～13 章)讲解了 Struts 2 与 Hibernate 的整合开发，Struts 2、Hibernate 和 Spring 的整合开发，以及 Struts 2 与 JFreeChart 的整合，还有 Struts 2 和 Ajax 的结合应用。最后实例篇(第 14～15 章)通过太极研修院企业网站和人力资源管理系统两个综合实例帮助读者全面掌握在实际项目中使用 Struts 2 技术，提高对大型应用系统的整体把握，使读者熟练掌握 Struts 2 技术。

本书适合具有一定 Web 开发经验的开发人员，或具有其他 Web 框架使用经验的开发人员或想要学习 Struts 2 开发的开发人员，以及正在从事 Java Web 开发的开发人员。

图书在版编目(CIP)数据

Struts 2 Web 开发学习实录/杨少敏，樊双灵编著. --北京：清华大学出版社，2011.7
(Web 程序员成功之路)
ISBN 978-7-302-25660-1

Ⅰ. ①S…　Ⅱ. ①杨…　②樊…　Ⅲ. ①软件工具—程序设计　Ⅳ. ①TP311.56

中国版本图书馆 CIP 数据核字(2011)第 092714 号

责任编辑：张　瑜
装帧设计：杨玉兰
责任校对：周剑云
责任印制：

出版发行：清华大学出版社　　**地　址：**北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编：**100084
社 总 机：010-62770175　　**邮　购：**010-62786544
投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015, zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　销：全国新华书店
开　本：190×260　**印　张：**39　**字　数：**941 千字
版　次：2011 年 67 月第 1 版　**印　次：**2011 年 7 月第 1 次印刷
印　数：1~4000
定　价：78.00 元

产品编号：

前　　言

对于 Struts 1 框架而言，由于与 JSP/Servlet 耦合非常紧密，因而导致了一些严重的问题。首先，Struts 1 支持的表现层技术单一。由于 Struts 1 出现的年代比较早，那个时候没有 FreeMarker、Velocity 等技术，因此它不可能与这些视图层的模板技术进行整合。其次，Struts 1 与 Servlet API 的严重耦合，使应用难于测试。最后，Struts 1 代码严重依赖于 Struts 1 API，属于侵入性框架。

Struts 2 的体系与 Struts 1 体系的差别非常大，因为 Struts 2 使用了 WebWork 的设计核心，而不是 Struts 1 的设计核心。Struts 2.0 其实就是 WebWork 2.3，如果读者使用过 WebWork 框架，那么学习 Struts 2 就能很快上手。Struts 2 中大量使用拦截器来处理用户的请求，从而允许用户的业务逻辑控制器与 Servlet API 分离。相信随着时间的推移，Struts 2 还将续写 Struts 1 的辉煌。因此，我们现在学习 Struts 2，可以提高我们的竞争力。

1．本书内容

第 1 章　Struts 2 扬帆起航。本章首先介绍了 Struts 2 的优点和框架架构，然后讲解了 Struts 2 的配置文件，最后介绍了 Struts 2 的标签库和控制器组件。

第 2 章　完美的 Struts 2 配置。本章首先向读者讲解了 Struts 2 的基本配置，如：web.xml、struts.xml、struts.properties、struts-default.xml 等，然后详细介绍了 Struts 2 的深入配置，接下来讲解了 Action 配置、Result 的配置、Result 的动态配置，最后介绍了 Struts 2 异常机制的应用。

第 3 章　数据类型大转换。本章首先介绍了类型转换的作用、如何使用类型转换器，以及 Struts 2 对 null 属性的处理，然后讲解了 Struts 2 的类型转换对 List、Map 和 Set 的支持，最后讲解了使用注解来配置类型转换。

第 4 章　国际化与异常处理。本章首先介绍了 Java 国际化的思路和 Struts 2 中的全局国际化资源文件，以及输出国际化消息，然后讲解了使用 Action 范围的国际化和使用<s:i18n/>标签实现国际化，最后讲解了使用 Struts 2 实现国际化。

第 5 章　Struts 2 中的拦路虎——拦截器。本章首先介绍了拦截器的配置，自定义拦截器的步骤以及配置，然后讲解运用方法过滤拦截器，并讲解了 Struts 2 的内置拦截器，最后讲解了拦截器注解操作，及完成权限控制拦截器。

第 6 章　探索数据校验的奥妙。本章首先介绍了在 Action 中通过编程对输入数据进行验证及 validateXxx()和 validate()方法的使用，然后介绍了 Struts 2 内置的验证器，以及验证框架在开发中的使用，最后介绍了开发自定义的验证器和验证注解的使用。

第 7 章　Struts 2 中完整的 OGNL。本章首先讲解了 OGNL 的三要素、OGNL 表达式的使用，然后讲解了 OGNL 对集合的操作、lambda 表达式的使用，最后讲解了 Struts 2 对 OGNL 表达式的增强。

第 8 章　Struts 2 的标签库。本章首先介绍了 Struts 2 的控制标签和数据标签的使用，然后讲解了模板和主题的应用以及表单标签的使用，最后介绍了非表单标签的使用。

第 9 章　轻松实现文件上传和下载。本章首先介绍了如何在 Struts 2 中实现文件上传，然后讲解了如何过滤文件上传的类型和大小，最后讲解了如何在 Struts 2 中实现文件下载和同时上传多个文件的步骤。

第 10 章　避免表单重复提交和等待页面。本章介绍了防止表单重复提交的机制、TokenIntercepter 拦截器的使用、TokenSessionStoreInterceptor 拦截器的使用以及使用 ExecuteAndWaitInterceptor 拦截器向用户显示等待页面。

第 11 章　黄金搭档——Struts 2 集成 Spring 与 Hibernate。本章首先介绍了 Hibernate 的作用以及 Hibernate 的开发应用，然后介绍了 Spring 的作用，最后详细讲解了 Struts 2 和 Hibernate 的集成开发以及 Struts 2、Hibernate 和 Spring 的集成开发。

第 12 章　整合 JFreeChart。本章首先介绍了 JFreeChart 生成饼图、柱状图以及折线图，然后讲解了 JFreeChart 生成时间顺序图和带交互功能的热点统计图，最后详细讲解了 Struts 2 与 JFreeChart 的整合。

第 13 章　当 Struts 2 碰见 Ajax。本章首先介绍了 Ajax 的输入校验，然后介绍了 DWR 框架的使用和 JSON 串作为数据的载体，最后讲解了 Dojo 框架的使用和 Struts 2 的 Ajax 标签。

第 14 章　太极研修院企业网站。本章使用 Struts 2 和 Hibernate 3 技术开发了一个太极研修院企业网站，实现的前台功能包括首页展示、企业简介、新闻中心、太极商城、在线视频、太极风采和培训招生等，其后台功能包括新闻中心、太极商城、信息管理、用户管理、日志管理和系统信息等。

第 15 章　人力资源管理系统。本章整体采用了 Struts 2、Hibernate 和 Spring 三大框架实现了一个人力资源管理系统。本系统具有员工管理、招聘管理、奖惩管理、培训管理、薪资管理及管理员模块。

2. 本书特色

本书中大量内容来自真实的 Struts 2 项目，力求通过读者实际操作中采取的解决问题的方法使读者更容易地掌握 Struts 2 的应用开发。本书难度适中，内容由浅入深，实用性强，覆盖面广，条理清晰。

1)　结构独特

通过“网络教学、基础知识、实例描述、实例应用、运行结果、实例分析”的形式，将每个知识与实际应用中的问题相结合。

2)　形式新颖

本书用准确的语言总结概念，用直观的图示演示过程，用详细的注释解释代码，用形象的比喻帮助记忆。

3)　技术文档

将一些非常简单的知识点或者理论性的内容穿插其中，通常这些文档没有具体的实践操作，但是读者又必须要了解的，像一些概念和术语。

4)　内容丰富

涵盖了实际开发中 Struts 2 技术所遇到的 Hibernate、Spring、JFreeChart 和 Ajax 等方面的热点问题。

5) 随书光盘

本书为实例配备了视频教学文件，读者可以通过视频文件更加直观地学习 Struts 2 的使用知识。

6) 网站支持

读者在学习或者工作的过程中，如果遇到实际问题，可以直接登录 www.itzcn.com 与我们取得联系，我们会在第一时间内给予帮助。

7) 贴心提示

为了便于读者阅读，全书还穿插了一些技巧、提示等小贴士，体例约定如下。

提示：通常是一些贴心的提醒，让读者加深印象或为读者提供建议，或者为读者提供解决问题的方法。

注意：提出学习过程中需要特别注意的一些知识点和内容，或者相关信息。

技巧：通过简短的文字，指出在应用知识点时的一些小窍门。

3．读者对象

本书具有知识全面、实例精彩、指导性强的特点，力求以知识的全面性及丰富的实例来指导读者透彻地学习 Struts 2 各方面的知识。本书可以作为 Struts 2 的入门书籍，也可以帮助中级读者提高技能，对高级读者也有一定的启发意义。

本书适合以下人员阅读学习。

- 具有一定 Web 开发经验的开发人员
- 具有其他 Web 框架使用经验的开发人员
- 想要学习 Struts 2 开发的开发人员
- 正在从事 Java Web 开发的开发人员

4．参编人员

本书主要由杨少敏、樊双灵编写，其他参与编写、资料整理、程序开发的人员还有杨伟康、岳培园、翟珊珊、张冬旭、赵振方、郑小营、赵俊昌、段韵治、胡海静等。

由于编者水平有限，书中难免存在不足和疏漏之处，恳请读者批评指正。

目　　录



第 1 章　Struts 2 扬帆起航

内容摘要：

Struts 只是一个 MVC 框架(Framework)，用于快速开发 Java Web 应用。Struts 实现的重点在 C(Controller)，包括 ActionServlet/RequestProcessor 和我们定制的 Action，也为 V(View)提供了一系列定制标签(Custom Tag)。但 Struts 几乎没有涉及 M(Model)，所以 Struts 可以采用 Java 实现的任何形式的商业逻辑。

Struts 最早是作为 Apache Jakarta 项目的组成部分问世运作。项目的创立者希望通过对该项目的研究，改进和提高 Java Server Pages、Servlet、标签库，提高面向对象的技术水准。

Struts 这个名字来源于在建筑和旧式飞机中使用的支持金属架。它的目的是减少在运用 MVC 设计模型来开发 Web 应用的时间。

Struts 跟 Tomcat、Turbine 等诸多 Apache 项目一样，是开源软件，这是它的一大优点，同时可以使开发者能更深入地了解其内部实现机制。

除此之外，Struts 的优点主要集中体现在两个方面：Taglib 和页面导航。Taglib 是 Struts 的标记库，灵活动用，能大大提高开发效率。另外，就目前国内的 JSP 开发者而言，除了使用 JSP 自带的常用标记外，很少开发自己的标记，或许 Struts 是一个很好的起点。

学习目标：

- 了解 Struts 的优点。
- 理解 Struts 2 的框架架构。
- 熟悉 Struts 2 的配置文件。
- 掌握 Struts 2 的标签库。
- 掌握 Struts 2 的控制器组件。

1.1　Struts 2 发展史

Struts 2 以 WebWork 优秀的设计思想为核心，继承了 Struts 1 的部分优点，建立了一个兼容 WebWork 和 Struts 1 的 MVC 框架。Struts 2 的目标就是希望可以和原来的 Struts 1、WebWork 的开发人员都可以平稳熟练地使用 Struts 2 的框架。

视频教学： 光盘/videos/01/ Struts_2_jianjie.avi　　**长度：7 分钟**

1.1.1　Struts 2 的简介

Struts 2 是 Struts 1 的下一代产品。它在 Struts 1 和 WebWork 的技术基础上进行了合并，是一种全新的框架。Struts 2 的体系结构与 Struts 1 有着巨大的差别。Struts 2 以 WebWork 为核心，采用拦截器的机制来处理用户的请求，这使得业务逻辑控制器能够与 Servlet API 完全脱离开，所以 Struts 2 可以理解为 WebWork 的更新产品。Struts 2 和 Struts 1 有着非常大的变化，但是相对于 WebWork，Struts 2 只有很小的变化。

Apache Struts 2 就是大家所熟知的 WebWork 2。它是一个优雅的、可扩展的 JAVA EE Web 框架。框架设计的目标贯穿于整个开发周期，从开发到发布，包括维护的整个过程。

在经历了几年的各自发展后，WebWork 和 Struts 社区决定合二为一，也即是 Struts 2。

1.1.2　Struts 2 和 Struts 1 的不同

Struts 2 与 Struts 1 已经不能再放到一起比较，虽然都是对 MVC 架构模式的实现，本质却完全不同。Struts 2 的前身是 WebWork，其实现方式和功能都要优于 Struts 1，但是，Struts 1 先入为主，很多应用程序都基于 Struts 1，其生命力和普及度使得 WebWork 落于下风。随着新思想和新架构的不断涌入，特别是 Web 2.0 被大量提及，Struts 1 显然无法跟上日新月异的变化，在很多应用上显得力不从心，最终催生了 Struts 2。可以说 Struts 2 是为变而变。

1. Action 类

- Struts 1 要求 Action 类继承于一个抽象基类。Struts 1 的一个普遍问题是使用抽象类编程而不是接口。
- Struts 2 的 Action 类可以实现一个 Action 接口，也可实现其他接口，使可选和定制的服务成为可能。Struts 2 提供一个 ActionSupport 基类去实现常用的接口。Action 接口不是必须的，任何有 execute 标识的 POJO 对象都可以作为 Struts 2 的 Action 对象。

2. 线程模式

- Struts 1 Action 是单例模式并且必须是线程安全的，因为仅有一个 Action 实例来处理所有的请求。单例策略限制了 Struts 1 Action 能做的事，并且要在开发时特别小心。因此 Action 资源必须是线程安全的或同步的。

- Struts 2 Action 对象为每一个请求产生一个实例，因此没有线程安全问题。实际上，Servlet 容器给每个请求产生许多可丢弃的对象，并且不会导致性能和垃圾回收问题。

3. Servlet 依赖

- Struts 1 Action 依赖于 Servlet API，因为当一个 Action 被调用时，HttpServletRequest 和 HttpServletResponse 被传递给 execute 方法。
- Struts 2 Action 不依赖于容器，允许 Action 脱离容器单独被测试。如果需要，Struts 2 Action 仍然可以访问初始的 request 和 response。但是，其他的元素减少或者消除了，有直接访问 HttpServetRequest 和 HttpServletResponse 的必要性。

4. 可测性

- 测试 Struts 1 Action 的一个主要问题是 execute 方法暴露了 servlet API(这使得测试要依赖于容器)。一个第三方扩展——Struts TestCase 提供了一套 Struts 1 的模拟对象(来进行测试)。
- Struts 2 Action 可以通过初始化、设置属性、调用方法来测试，“依赖注入”支持使测试更加容易。

5. 表达式语言

- Struts 1 整合了 JSTL，因此使用 JSTL EL。这种 EL 有基本对象图遍历，但是对集合和索引属性的支持很弱。
- Struts 2 可以使用 JSTL，它更支持一个更强大和灵活的表达式语言——Object-Graph Navigation Language(OGNL 对象图导航语言)。
- Struts 1 使用标准 JSP 机制把对象绑定到页面中来访问。
- Struts 2 使用 ValueStack 技术，使 taglib 能够访问值而不需要把页面(view)和对象绑定起来。ValueStack 策略允许通过一系列名称相同但类型不同的属性重用页面(view)。

6. 类型转换

- Struts 1 ActionForm 属性通常都是 String 类型。Struts 1 使用 Commons-Beanutils 进行类型转换。每个类一个转换器，对每一个实例来说是不可配置的。
- Struts 2 使用 OGNL 进行类型转换。由他(OGNL)提供基本和常用对象的转换器。

7. 校验

- Struts 1 支持在 ActionForm 的 validate()方法中手动校验，或者通过 Commons Validator 的扩展来校验。同一个类可以有不同的校验内容，但不能校验子对象。
- Struts 2 支持通过 validate()方法和 XWork 校验框架来进行校验。XWork 校验框架使用为属性类类型定义的校验和内容校验，来支持 chain 校验子属性。

8. Action 执行的控制

- Struts 1 支持每一个模块，有单独的 Request Processors(生命周期)，但是模块中的所有 Action 必须共享相同的生命周期。

- Struts 2 支持通过拦截器堆栈(Interceptor Stacks)为每一个 Action 创建不同的生命周期。堆栈能够根据需要和不同的 Action 一起使用。

1.2　Struts 2 体系介绍

对于学过 Struts 1 体系的人来说，在下面的学习中会感觉 Struts 1 和 Struts 2 的体系差别非常大，因为 Struts 2 使用了 WebWork 的设计核心，而不是使用了 Struts 1 的设计核心。Struts 2 使用大量的拦截器来处理用户的请求，允许用户的业务逻辑控制器来与 Servlet API 分离。

1.2.1　基础知识——Struts 2 框架架构流程

一个请求在 Struts 2 框架中的处理大概分为以下几个步骤。

(1) 客户端提交一个 HttpServletRequest 请求，例如在浏览器中输入 http://localhost:8080/Struts2/ch1/Reg.action 就是提交一个 HttpServletRequest 请求。

(2) 请求被提交到一系列(主要是 3 层)的过滤器(Filter)，如 ActionContextCleanUp、SiteMesh 和 FilterDispatcher 等。



注意这里是有顺序的，首先是提交 ActionContextCleanUp，再到其他过滤器(Othter Filters、SiteMesh 等)，最后到 FilterDispatcher。

(3) FilterDispatcher 是控制器的核心，就是 MVC 的 Struts 2 实现中控制层的核心。FilterDispatcher 询问 ActionMapper 是否需要调用某个 Action 来处理这个请求，如果 ActionMapper 决定需要调用某个 Action，FilterDispatcher 则把请求的处理交给 ActionProxy。

(4) ActionProxy 通过 Configuration Manager(struts.xml)询问框架的配置文件，找到需要调用的 Action 类。例如，用户注册示例将找到 UserReg 类。ActionProxy 创建一个 ActionInvocation 实例，同时 ActionInvocation 通过代理模式调用 Action。但在调用之前，ActionInvocation 会根据配置加载 Action 相关的所有 Interceptor(拦截器)。一旦 Action 执行完毕，ActionInvocation 负责根据 struts.xml 中的配置找到对应的返回结果 result。图 1-1 是 Struts 流程图。

1.2.2　基础知识——Struts 2 的配置文件

Struts 2 共有 5 类配置文件，分别是 struts.xml、struts.properties、Web.xml、struts-plugin.xml 和 struts_default.xml。本节将详细向大家讲解这些配置文件的相关知识。

1．struts.xml 文件

struts.xml 定义应用自身使用的 action 映射及 result，但我们一般将应用的各个模块分到不同的配置文件中。

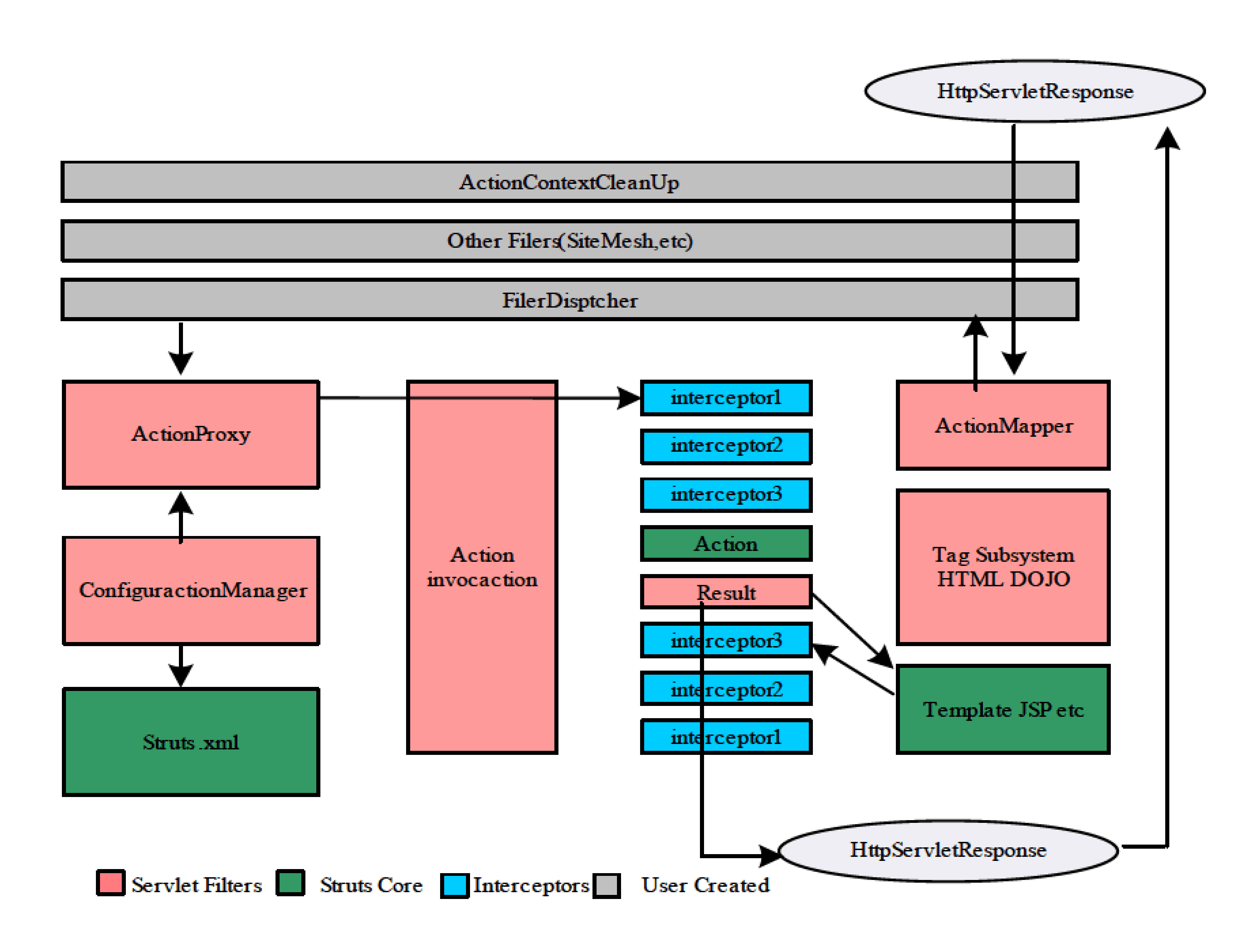


图 1-1　Struts 流程图

首先必须在 struts.xml 文件的对应程序做相应的配置，其中需要对每个动作进行相应拦截器的调用，对每种动作的运行结果进行配置等。在拦截器中，必须在使用前进行注册，Struts 配置文件可以支持继承，默认的配置文件包在 Struts 2-core-VERSION.jar 中。struts-default.xml 文件是那些默认配置文件之一。它的主要功能就是用来注册默认的结果类型和拦截器。所以，在使用的时候没必要在 struts.xml 文件里进行注册，就可以使用默认的结果类型和拦截器。

struts.xml 文件代码如下所示。

```
<!-- 省略部分代码 -->
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.1.7//EN"
    "http://struts.apache.org/dtds/struts-2.1.7.dtd">
<struts>
    <constant name="struts.i18n.encoding" value="gbk" />
    <include file ="struts-default.xml" />
    ......
</struts>
<!-- 省略部分代码 -->
```

在 struts 节点下有很多的子节点，而这些子节点则是配置框架的主要因素。接下来讲解 struts 节点下的子节点的含义。

1) include 元素

在 strut.xml 中我们会经常发现 include 元素。一个大的应用程序可能有许多的包，为了使 strut.xml 便于管理，我们可以把它拆分成几个小的配置文件，然后再使用 include 元素来应用这些拆分的配置文件。

2) package 元素

为了方便使用，Struts 可以把各种动作分门别类地组织成不同的包，即 package。在此可以把它当作 struts.xml 文件典型的模板。详细代码如下所示。

```
<!-- 省略部分代码 -->
    <package name="userinfo" namespace="/userinfo" extends="struts-default">
            <global-results>
            <result name="error">/exception.jsp</result>
        </global-results>
        <global-exception-mappings>
            <exception-mapping result="error"
                exception="java.lang.RuntimeException">
            </exception-mapping>
        </global-exception-mappings>
        <action name="loginuser" class="com.itzcn.action.Userinfo">
          <result name="loginout">/loginout.jsp</result>
        </action>
    </package>
<!-- 省略部分代码 -->
```

在上述配置文件中，package 可以有一个或多个，每个 package 都必须有一个不同的 name 属性。其中 namespace 属性是可有可无的属性，如果没有设置该属性则默认是“/”，如果设置了该属性的值，那么在使用这个包里面的动作时，就必须把该属性的命名空间添加到有关的 URI 字符串中。global 节点是用来捕获异常的，在设置了 java.ang.RuntimeException 后运行出现异常，这样捕捉到异常之后转向指定的 error 页面。

3) constant 元素

如果不需要创建一个新的文件，那么可以在 struts.xml 文件里使用 constant 元素。详细代码如下所示。

```
<!-- 省略部分代码 -->
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name="struts.i18n.encoding" value="gbk" />
    <constant name="struts.devMode" value="true" />
</struts>
<!-- 省略部分代码 -->
```

其中 struts.xml 文件主要负责管理应用中的 Action 映射以及该 Action 包含的 Result 定义等。除此之外，Strut 2 框架还包含了一个 struts.properties 文件，该文件定义了 Struts 2 框架的大量

属性，可以通过修改这些属性来满足我们的需求。

2. struts.properties 文件

struts.properties 用来定义框架自身的全局变量，该文件定义的全局属性也可以在 struts.xml 中定义。

struts.properties 文件是一个标准的 Properties 文件，该文件包含了一系列的 key-value 对象，每个 key 就有一个 Struts 2 属性，该 key 对应的 value 就是一个 Struts 2 属性值。该文件通常放在 Web 应用的 WEB-INF/classes 路径下。将该文件放入 Web 应用程序下的 CLASSPATH 路径下，Struts 2 框架就可以加载该文件。

3. Web.xml 文件

准确地说，Web.xml 不属于 Struts 2 框架特有的文件。作为部署文件，Web.xml 是所有 Java Web 项目的核心文件。然而在这里之所以讲到该配置文件，是因为在使用 Struts 2 框架时，需要在 Web.xml 中配置一个前段控制器 FilterDispatcher，用于对 Struts 2 框架进行初始化和处理所有的请求。详细代码如下所示。

```
<!-- 省略部分代码 -->
<?xml version="1.0" encoding="UTF-8"?>
<Web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/Web-app 2 5.xsd">
    <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
        <filter>
            <filter-name>struts2</filter-name>

    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
        </filter>
    <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>*.action</url-pattern>
</filter-mapping></Web-app>
<!-- 省略部分代码 -->
```

4. struts-plugin.xml 文件

struts-plugin.xml 文件是 struts 插件使用的配置文件，例如当结合 struts 和 spring 一起使用时就需要在 Web.xml 中引用该配置文件，这里不再详细说明。

5. struts-default.xml 文件

struts-default.xml 用来定义框架自身使用的 action 映射及 result 定义，是 struts 2 框架默认加载的配置文件。它定义了 struts 2 的一些核心 bean 和拦截器。struts-default 包就是在 struts-

default.xml 文件中定义的。该文件中还可以定义 Struts 2 内置的结果类型、内置的拦截器以及不同拦截器组成的拦截器栈，这些拦截器栈可以直接使用。在 strut-default.xml 文件的最后还可以定义默认的拦截器引用。



如果让 Struts 2 不加载 struts-default.xml，或者加载自定义的配置文件，可以在 struts.properties 文件中设置 struts.configuration.files 属性。

1.2.3 基础知识——Struts 2 的控制器组件

Struts 2 的控制器组件是 Struts 2 框架的核心，事实上，所有 MVC 框架都是以控制器组件为核心的。Struts 2 的控制器由两部分组成：FilterDispatcher 和业务控制器 Action。

Struts 2 应用中起作用的业务控制器不是用户定义的 Action，而是系统生成的 Action 代理，但该 Action 代理是以用户定义的 Action 为目标的。详细代码如下所示。

```
<!-- 省略部分代码 -->
import com.opensymphony.xwork2.ActionSupport;
public class Userinfo extends ActionSupport{
    private String username;
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String login(){
        String num="";
        if(username.equals("admin"))
        {
            num="loginout";
        }
        return num;
    }
}
<!-- 省略部分代码 -->
```

通过查看上面的 Action 代码，发现该 Action 比 WebWork 中的 Action 更彻底。该 Action 无需实现任何父接口，无需继承任何 Struts 2 基类，该 Action 类完全是一个 POJO(普通、传统的 Java 对象)，因此具有很好的复用性。

1.2.4 实例描述

以上的讲解只是大致使读者了解了 Struts 2 的基本应用原理，接下来将使用一个简单的会员登录系统来检验一下对 Struts 2 的掌握情况。通过 Struts 2 的配置文件对系统进行配置，获取用户输入的用户名和密码，然后判断用户是否登录成功！

1.2.5　实例应用

【例 1-1】 Struts 2 体系介绍。

在开始制作项目之前，首先需要将开发环境搭建好。本章主要讲解开发环境和运行环境的搭建。表 1-1 所示是搭建开发环境所需要的软件，下载后进行安装，配置环境。

表 1-1　环境配置软件

软　件	版　本	下载地址
JDK	1.6.0	http://java.sun.com
Tomcat	6.0	http://tomcat.apache.org
MyEclipse	8.5	http://downloads.myeclipseide.com/

首先在 MyEclipse 开发工具中，创建一个名为 News 的 Web 项目，然后进行环境搭建。

1．硬件环境

- 处理器：Intel Pentium。
- 内存：32M 或以上。
- 硬盘空间：1GB 以上。

2．软件环境

- 操作系统：Windows 2003 /2000/XP。
- Web 服务器：Tomcat 6.0 或以上版本。
- 开发工具：MyEclipse。
- 客户端：IE 5.0 或以上版本。
- 开发语言：JSP、Java。

3．配置环境

首先将 JAR 包导入工程的 lib 目录下，需要导入 commons-fileupload-1.2.1.jar、commons-logging-1.0.4.jar、freemarker-2.3.15.jar、ognl-2.7.3.jar、Struts 2-core-2.1.8.1.jar、xwork-core-2.1.6.jar 六个 JAR 包。

1)　定义 struts.xml 文件

接下来在 src 目录下创建一个 struts.xml 配置文件，代码如下所示。

```
<!-- 省略部分代码 -->
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.1.7//EN"
    "http://struts.apache.org/dtds/struts-2.1.7.dtd">
<struts>
    <constant name="struts.i18n.encoding" value="gbk" />
    <include file ="struts-default.xml" />
```

```
    <package name="userinfo" namespace="/" extends="struts-default">
        <global-results>
            <result name="error">/error.html</result>
        </global-results>
        <global-exception-mappings>
            <exception-mapping result="error"
                exception="java.lang.RuntimeException">
            </exception-mapping>
        </global-exception-mappings>
        <action name="loginuser" class="com.itzcn.action.Userinfo">
        <result name="loginout">/loginout.jsp</result>
        <result name="index">/index.jsp</result>
        </action>
    </package>
</struts>
<!-- 省略部分代码 -->
```

在上述配置文件中，通过配置 constant 节点进行编码格式。在 struts.xml 文件中，配置运行时异常处理的页面请求。在该配置文件中，最重要的是 action 的配置，在此指定 action 的业务类地址和请求转发的页面。

2) 配置 Web.xml 文件

配置好 struts.xml 文件后则是配置 Web.xml 文件。在 Web.xml 文件中配置一个前端控制器 FilterDispatcher，用于对 Struts 框架的初始化和处理所有的请求。详细代码如下所示。

```
<!-- 省略部分代码 -->
<?xml version="1.0" encoding="UTF-8"?>
<Web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/Web-app_2_5.xsd">
    <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <filter>
        <filter-name>struts2</filter-name>

    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
    </filter>
    <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>*.action</url-pattern>
    </filter-mapping></Web-app>
<!-- 省略部分代码 -->
```

在上述配置文件中，主要配置了 filter 节点 Struts 2 的过滤器和接受请求的方式。

3)　编写 Action 类

接下来编写业务处理的 Action，在编写 Action 之前，首先要明白 Action 的主要作用，它主要的功能是对用户提交过来的内容信息作出相应的回复。Action 和普通的类一样，只是它继承了一个 ActionSupport 类。详细代码如下所示。

```
<!-- 省略部分代码 -->
public class Userinfo extends ActionSupport{
    private String txtUsername;
    private String txtPass;
    public String getTxtUsername() {
        return txtUsername;
    }
    public void setTxtUsername(String txtUsername) {
        this.txtUsername = txtUsername;
    }
    public String getTxtPass() {
        return txtPass;
    }
    public void setTxtPass(String txtPass) {
        this.txtPass = txtPass;
    }
    public String logi()
    {
        String num="index";
        if(txtUsername.equals("admin")&&txtPass.equals("admin"))
        {
            num="loginout";
        }
        HttpServletRequest httprequest = ServletActionContext.getRequest();
        httprequest.setAttribute("ms","登录失败！密码错误！");
        return num;
    }
}
<!-- 省略部分代码 -->
```

在上述代码中，定义了两个成员变量和它们的 GET()和 SET()方法。要注意的是该变量名称必须和页面中表单提交的值名称一样。通过判断返回相应的页面，在这里判断用户名和密码是否等于“admin”。

1.2.6　运行结果

通过以上的代码编写和环境搭建配置，接下来将是显现结果的时刻。运行结果如图 1-2 所示。

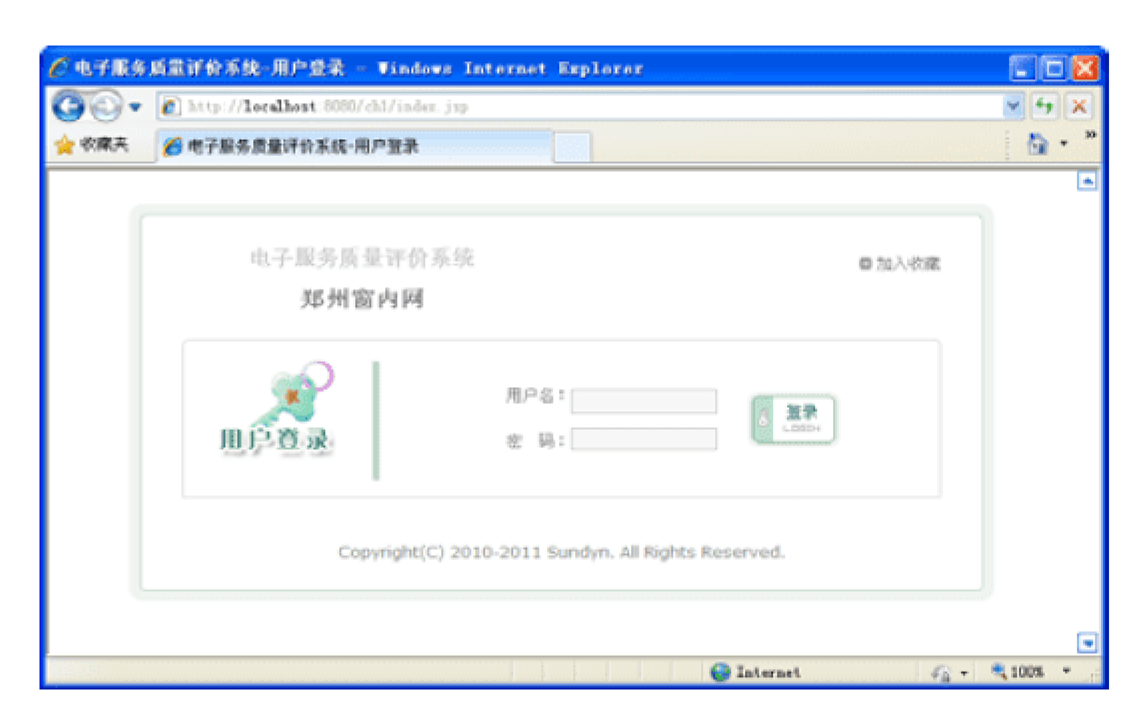


图 1-2　用户登录系统

1.2.7　实例分析



源码分析：

在上述例子中，主要学习的内容就是 Struts 2 框架的搭建和整合。在搭建该框架过程中，最为重要的是它的配置，其中 struts.xml 文件是 Struts 2 中的核心配置文件，它是 Struts 2 框架使用的核心组件。Bean 包含的静态方法需要一个值注入。它可以在不创建某个类实例的情况下接收框架常量。通常需要设置 static= “true”。当指定了 type 属性时，该属性不应该指定为“true”。

1.3　Struts 2 的 Hello World

“Hello World”程序指的是只在计算机屏幕上输出“Hello World!”。一般来说，这是计算机编程语言中最基本、最简单的程序，也通常是初学者编写的第一个程序。通过这个程序，开发者可以确定针对该语言的编译器、程序开发环境，以及运行环境是否安装正确。在此也可以用 Struts 2 来实现一个“Hello World”的程序。



视频教学：光盘/videos/01/ Struts2_develop.avi　　　　长度：8 分钟

1.3.1　基础知识——Struts 2 标签

Struts 2 标签库提供了非常丰富的功能，因此 Struts 2 标签库也是 Struts 2 中最重要的一部分，这些标签不仅提供了表示层的数据显示处理，而且还提供了基本的流程控制功能，同时还支持国际化和 Ajax 等功能。

之所以使用 Struts 2 标签，是因为这些标签可以为开发者减少大量的代码书写，而且使用也非常方便。

在 JSP 页面中引用标签库，需要使用 taglib 指令。该指令的 uri 属性的值设置为<uri>元素的内容，prefix 属性则设置为该标签的标题，通常设置为“s”。详细代码如下所示。

```
<%@ taglib prefix="s" uri="/struts-tags" %>
```



Struts 2 标签库的功能非常复杂，该标签库可以完全替代 JSTL 标签库，而且 Struts 2 的标签支持表达式的语言。

1.3.2　实例描述

该实例显现的是：通过用户输入不同用户的名称，使程序输出 Hello World 字符串。首先创建一个 Web 项目，名字为 Hello World，详细结构如图 1-3 所示。然后，将需要导入的 JAR 包导入工程的 lib 目录下，通过用户在页面中输入自己的用户名，单击“确定”按钮进入显示信息页面。

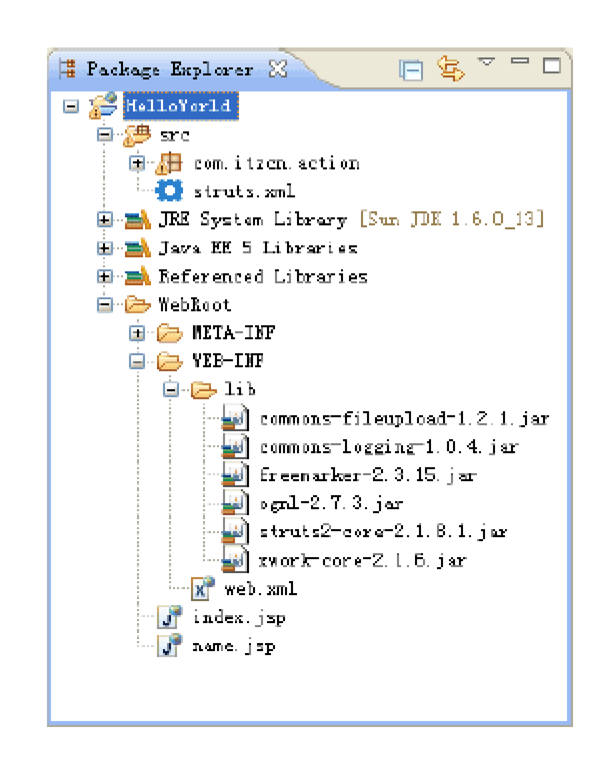


图 1-3　工程结构图

1.3.3　实例应用

【例 1-2】 Struts 2 的 Hello World。

创建好项目后，需要做的第二步则是将项目搭建上 Struts 2 的运行环境，首先需要配置 struts.xml 文件。详细代码如下所示。

```
<!-- 省略部分代码 -->
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.1.7//EN"
    "http://struts.apache.org/dtds/struts-2.1.7.dtd">
<struts>
    <constant name="struts.i18n.encoding" value="gbk" />
     <include file ="struts-default.xml" />
     <package name="test" namespace="/" extends="struts-default">
         <global-results>
```

```
            <result name="error">/error.html</result>
        </global-results>
        <global-exception-mappings>
            <exception-mapping result="error"
                exception="java.lang.RuntimeException">
            </exception-mapping>
        </global-exception-mappings>
        <action name="helloworld" class="com.itzcn.action.HelloWorld">
          <result name="index">/index.jsp</result>
        </action>
    </package>
</struts>
<!-- 省略部分代码 -->
```

在配置文件中，配置了一个 Hello World 的 Action，它的 Class 文件是 HelloWorld 类。在 action 节点中 result 属性用来配置转发页面，即执行成功后的页面地址。

接下来配置 Web.xml 文件，这里主要配置 filter 节点 Struts 2 的过滤器和接受请求的方式，设置 filter-class 为 org.apache.Struts 2.dispatcher.FilterDispatcher。详细代码如下所示。

```
<!-- 省略部分代码 -->
<?xml version="1.0" encoding="UTF-8"?>
<Web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/Web-app 2 5.xsd">
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
    <filter>
       <filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
    </filter>
  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>*.action</url-pattern>
  </filter-mapping></Web-app>
<!-- 省略部分代码 -->
```

配置好 Web.xml 后， Struts 2 框架就搭建完成了一部分。接下来进行业务处理阶段，即将用户名提交到后台，经过处理后返回到前台页面。详细代码如下所示。

```
<!-- 省略部分代码 -->
public class HelloWorld extends ActionSupport{
    private String username;
    public String getUsername() {
        return username;
    }
```

```
    public void setUsername(String username) {
        this.username = username;
    }
    public String out()
    {
        HttpServletRequest httprequest = ServletActionContext.getRequest();
        httprequest.setAttribute("ms",username+"说: "Hello World"");
        return "index";
    }
}
<!-- 省略部分代码 -->
```

在上述代码中，声明了一个 username，用来保存用户 form 提交的用户名信息，调用 out()方法，将显示的信息保存到 request 对象里面，最后返回页面。页面显示信息代码如下所示。

```
<!-- 省略部分代码 -->
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>信息</title>
  </head>

  <body>
   <s:property value="#request.ms"/>
  </body>
</html>
<!-- 省略部分代码 -->
```

在页面应用标签库之前，需要将标签库引入到页面内。通过 property 标签使信息显示在前台页面内。

1.3.4　运行结果

以上的实例代码运行后的效果如图 1-4 和图 1-5 所示。

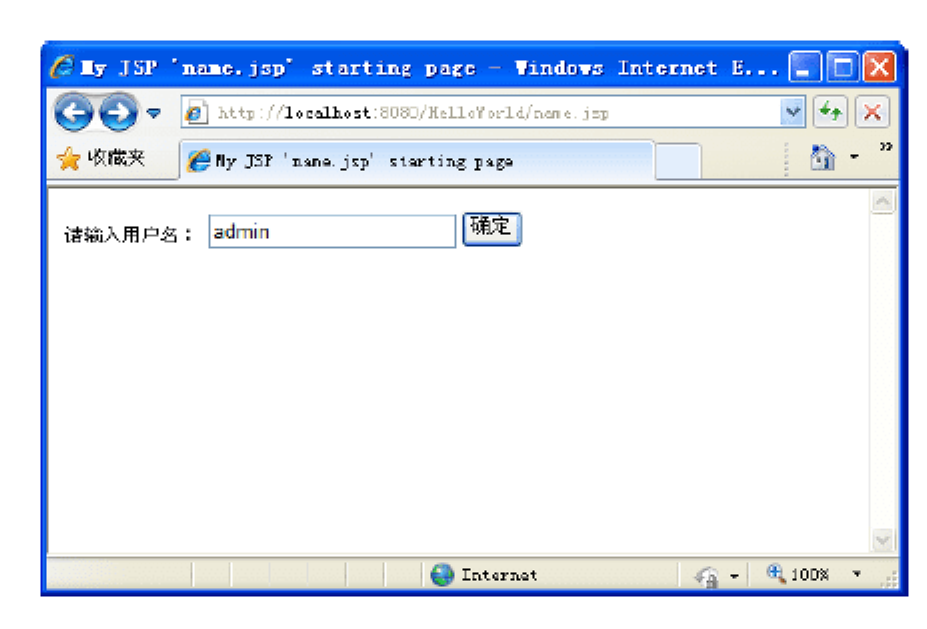


图 1-4　用户名输入界面

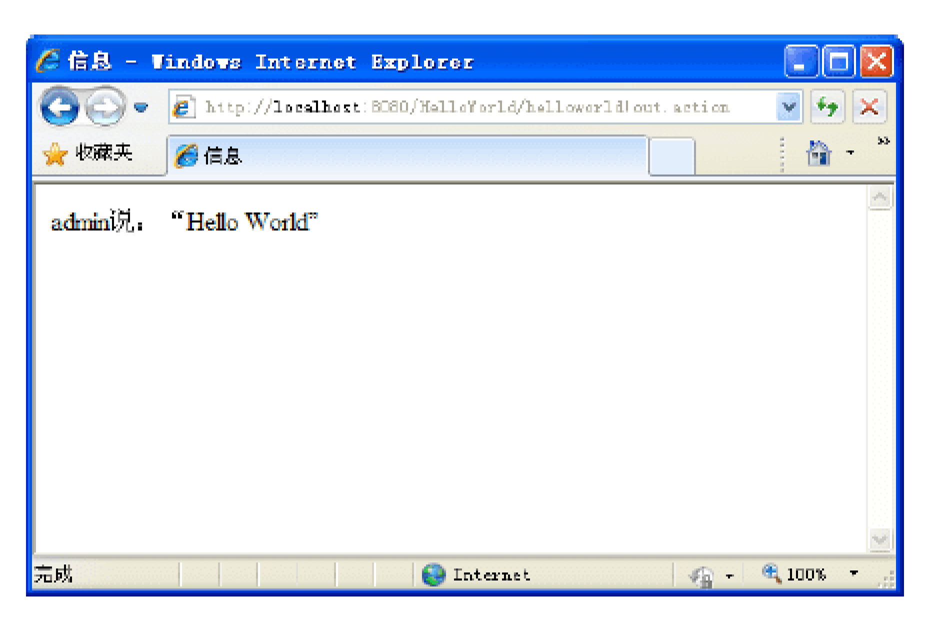


图 1-5　显示内容页面

1.3.5 实例分析



源码分析：

本节以一个 Hello World 应用为例，简要介绍了 Struts 2 MVC 框架的基本流程，从 Action 类基础的流程控制讲起，详细介绍了如何开发一个 Struts 2 应用。在该实例后面部分，在基本的 Struts 2 应用基础上，介绍了一些 Struts 2 的深入应用，包括在 Action 中访问 HttpSession 状态，将 Action 处理的结果显现在前台 JSP 页面内。同时还讲解了 Struts 2 的标签库。通过以上实例使读者对 Struts 2 有了进一步的了解。

1.4 常见问题解答

1.4.1 配置 struts.xml 时，class 路径错误



配置 struts.xml 文件时，class 路径错误！
网络课堂：http://bbs.itzcn.com/thread- 10922-1-1.html

在配置 struts.xml 文件，进行配置 Action 时，经常在启动 Tomcat 会出现错误信息，如图 1-6 所示。

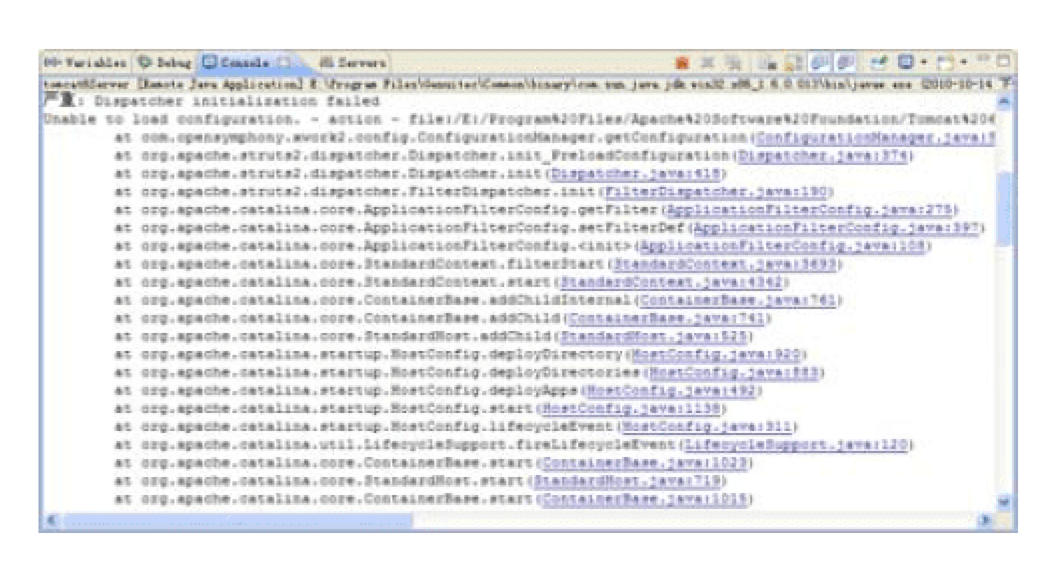


图 1-6　运行错误

【解决办法】该错误出现的原因多数是因为在配置 Action 节点时该节点的 class 元素路径错误，或者该路径不存在。

1.4.2 Struts 2 标签库引用错误



Struts 2 标签库引用错误！
网络课堂：http://bbs.itzcn.com/thread-10927-1-1.html

写一个项目案例，环境搭建 OK，所有的工作都准备就绪后，开始套页面显示内容信息，可是当将 Struts 2 标签库引入到需要的 JSP 页面时，总是出现“Can not find the tag library

descriptor for "/struts-tags"”这样的错误警告。

【解决办法】 经过调查，发现这个问题的主要原因在于没有标签库，缺少类库，没有导入 Struts 2 中标签库最重要的一个 JAR 包 Struts 2-core-2.1.8.1.jar。所以读者以后在使用 Struts 2 时一定要注意此类问题。

1.5　习　　题

一、填空题

(1) 在搭建 Struts 2 时，我们需要导入 commons-fileupload-1.2.1.jar、commons-logging-1.0.4.jar、freemarker-2.3.15.jar、ognl-2.7.3.jar、xwork-core-2.1.6.jar 和__________到 lib 目录下。

(2) Struts 2 核心配置文件是__________。

(3) Struts 2 中的 struts.xml 配置文件 action 节点的作用是__________。

(4) 若将一个普通的类转换成 Action 类，只要继承__________类即可。

二、选择题

(1) 以下标签，不属于 Struts 2 中的标签的是__________。

A. set 的标签　　B. append 标签

C. gererator 标签　　D. if 标签

(2) Struts 2 中不属于 Struts 2 的主要配置文件的是__________。

A. Web.xml 配置文件　　B. struts.xml 配置文件

C. struts.properties 配置文件　　D. applicationContext.xml 配置文件

(3) 下属选项中，不属于 strut.xml 配置文件中的元素的是__________。

A. package 元素　　B. file 元素

C. include 元素　　D. action 元素

三、上机练习

上机练习：使用 Struts 2 实现用户注册功能。

要求使用 Struts 2 实现用户注册功能，当用户单击“提交”按钮之后将用户填写的用户信息保存到数据库内。

运行页面后，要在页面内显示实体的属性信息，效果如图 1-7 所示。



图 1-7　用户注册



第 2 章　完美的 Struts 2 配置

内容摘要：

Struts 2 框架的配置文件分为两类：内部使用和供开发人员使用。内部配置文件由 Struts 2 框架自动加载，对其自身进行配置。其余的配置文件由开发人员使用，用于对 Web 应用进行配置。本章主要围绕 Struts 2 中的各项配置进行介绍。如 Struts 2 的相关配置文件包括 web.xml、struts.xml 和 struts.properties 文件等，对于开发人员，主要掌握 struts.xml 文件的配置即可。同时重点介绍了 struts.xml 文件的配置包含 Bean 配置、常量配置、包配置、命名空间配置等。讲解了 Struts 2 的核心——Action 的实现和配置，还有与 Action 紧密相关的 Result 配置，以及当 Action 处理请求结束时，系统下一步所要呈现的结果。最后还简要介绍了动态结果的使用以及 Struts 2 异常机制的应用。

学习目标：

- 熟悉 Struts 2 的基本配置。
- 掌握 Struts 2 的深入配置。
- 熟悉 Action 的配置。
- 熟悉 Result 的配置。
- 理解 Result 的动态配置。
- 掌握 Struts 2 异常机制的应用。

2.1 小小图书馆

要掌握 Struts 2 的基本配置，读者不要去死记硬背，只需大致浏览一遍基础知识，有一些初步印象后，再结合本节的实例应用，加深理解各种配置的具体应用即可。

视频教学： 光盘/videos/02/config.avi　　长度：7 分钟
光盘/videos/02/ web.xml.avi　　长度：10 分钟
光盘/videos/02/ struts.xml.avi　　长度：13 分钟
光盘/videos/02/ struts.properties.avi　　长度：5 分钟
光盘/videos/02/ struts-default.xml.avi　　长度：7 分钟
光盘/videos/02/ struts-plugin.xml.avi　　长度：5 分钟

2.1.1 基础知识——Struts 2 的基本配置

本节主要介绍 Struts 2 的各种配置文件，如 struts.xml、struts.properties 等，同时也介绍了 Struts 2 在 web.xml 文件中的配置。重点要掌握的是 web.xml、struts.xml 文件中的各项配置。掌握了配置文件的用法，才能更好地使用和扩展 Struts 2 框架的功能。

1. web.xml

准确地说，web.xml 并不是 Struts 2 框架特有的文件。作为部署描述符文件，web.xml 是所有 Java Web 应用程序都需要的核心配置文件，起着初始化 Servlet、Filter 等 Web 程序的作用。

对于 Struts 2 框架而言，需要在 web.xml 文件中配置一个前端控制器——FilterDispatcher，用于对 Struts 2 框架初始化和处理所有的请求。

配置 FilterDispatcher 的代码片段如下所示。

```
<!-- 配置 Struts 2 框架的核心 Filter -->
<filter>
    <!-- 配置 Struts 2 核心 Filter 的名字 -->
    <filter-name>struts</filter-name>
    <!-- 配置 Struts 2 核心 Filter 的实现类 -->
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher
</filter-class>
    <init-param>
    <!-- 配置 Struts 2 框架默认加载的 Action 包结构 -->
        <param-name>actionPackages</param-name>
        <param-value>org.apache.struts2.showcase.person</param-value>
    </init-param>
    <!-- 配置 Struts 2 框架的配置提供者类 -->
    <init-param>
        <param-name>configProviders </param-name>
        <param-value>lee.MyConfigurationProvider</param-value>
    </init-param>
</filter>
```

```
<!-- 配置 Filter 拦截的 URL-->
<filter-mapping>
    <!-- 配置 Struts 2 的核心 FilterDispatcher 拦截所有用户请求 -->
    <filter-name>struts</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

FilterDispatcher 可以包含一些初始化的参数，如下所示。

1) config

要加载的 XML 配置文件的列表(以逗号分隔)。如果没有设置 config 这个参数，Struts 2 框架默认将加载 struts-default.xml、struts-plugin.xml 和 struts.xml 这三个文件。

2) actionPackages

以逗号分隔的 Java 包名的列表，Struts 2 框架将扫描这些包中的 Action 类。

3) configProviders

实现了 ConfigurationProvider 接口的 Java 类的列表(以逗号分隔)。如果用户需要实现自己的 ConfigurationProvider 类，用户可以提供一个或多个实现 ConfigurationProvider 接口的类，然后将这些类的类名设置成该属性的值，多个类名之间以英文逗号(,)隔开。

4) *

任何其他的参数被当作是 Struts 2 的常量。每个<init-param>元素配置一个 Struts 2 常量，其中<param-name>子元素指定了常量 name，而<param-value>子元素指定了常量 value。<filter-mapping>元素是过滤器(Filter)必须的一个元素，用于过滤请求的路径，此处一般设为/*形式，对所有请求 uri 进行拦截(过滤)。

配置完 Struts 2 的核心控制器后，基本完成了 Struts 2 在 web.xml 文件中的配置。



如果 web 容器是 J2EE1.3(servlet2.3)，由于不会自动加载 struts 的标签库，所以需要在 web.xml 文件中手动加载 struts 的标签库，文件名 struts-tags.tld，一般放在 WEB-INF 下面，可以自己指定。但如果 web 容器是 J2EE1.4(servlet2.4)，那么 web 容器会自动加载标签库，Struts 2 的标签库定义文件包含在 struts2-core-2.0.6.jar 文件里，在 struts2-core-2.0.6.jar 文件的 META-INF 路径下，包含了一个 struts-tag.tld 文件。

2. struts.xml

struts.xml 文件是整个 Struts 2 框架的核心。struts.xml 文件内定义了 Struts 2 的系列 Action。定义 Action 时，指定该 Action 的实现类，并定义该 Action 处理结果与视图资源之间的映射关系。

下面将讲解 struts.xml 文件的具体配置。该示例配置代码如下。

```
<?xml version="1.0" encoding="UTF-8" ?>                     <!--XML 文件标识-->
<!DOCTYPE struts PUBLIC                                     <!--文档类型声明-->
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name="struts.i18n.reload" value="false" />
    <constant name="struts.custom.i18n.resources"
value="globalMessages"/>
    <include file="struts-chat.xml" />                      <!--包含其他 XML 文件-->
```

```
    <include file="struts-validation.xml" />
    <include file="struts-continuations.xml"/>
    <include file="struts-tags.xml"/>
    <!-- 配置包元素,该元素可以出现一次或者多次 -->
    <package name="default" extends="struts-default"> <!--配置 default 包-->
        <interceptors>                                <!--拦截器根元素-->
            <interceptor-stack name="crudStack">      <!--拦截器栈-->
                <interceptor-ref name="checkbox" />   <!--拦截器-->
                <interceptor-ref name="params" />
                <interceptor-ref name="static-params" />
                <interceptor-ref name="defaultStack" />
            </interceptor-stack>
        </interceptors>
        <action name="showcase">                      <!--配置 Action-->
            <result>showcase.jsp</result>             <!--配置 Result-->
        </action>
    </package>
    <!-- 配置 employee 包,并且指定命名空间为 employee -->
    <package name="employee" extends="default" namespace="/employee">
        <default-interceptor-ref name="crudStack"/>   <!--配置默认拦截器-->
        <action name="list"class="org.apache.struts2.showcase.action.
        EmployeeAction" method="list">
            <result>/empmanager/listEmployees.jsp</result>
            <interceptor-ref name="basicStack"/>
        </action>
    </package>
</struts>
```

在该配置文件中，有很多关键属性需要读者掌握它们的特性，简介如下。

1) include

include 节点是 Struts 2 中组件化的方式，可以将每个功能模块独立到一个 XML 配置文件中，然后用 include 节点引用。下面的代码就是利用 include 引入 struts-default.xml 文件。

```
    <include file="struts-default.xml"></include>
```

2) package

package 提供了将多个 Action 组织为一个模块的方式。package 的名字必须是唯一的，package 可以扩展。当一个 package 扩展自另一个 package 时，该 package 会在本身配置的基础上加入扩展的 package 配置。父 package 必须在子 package 前配置。可以为 package 配置如下属性。

- name：package 名称。
- extends：继承的父 package 名称。
- abstract：设置 package 的属性为抽象的，抽象的 package 不能定义 Action。
- namespace：定义 package 命名空间，该命名空间将影响 URL 地址。例如此命名空间为/test，那么访问是的地址为“http://localhost:8080/struts2/test/XX.action”。



由于 struts.xml 文件是自上而下解析的，所以被集成的 package 要放在集成 package 的前边。

下面的代码配置了一个名称为 com.kay.struts2 的包，该包继承于 struts-default 文件，该包的命名空间为“/test”：

```
    <package name="com.kay.struts2" extends="struts-default"
namespace="/test">
```

3)　定义拦截器

定义拦截器，可以配置如下属性。

- name：拦截器名称。
- class：拦截器类路径。

```
<interceptors>
    <interceptor name="timer" class="com.kay.timer"></interceptor>
    <interceptor name="logger" class="com.kay.logger"></interceptor>
    <!-- 定义拦截器栈 -->
    <interceptor-stack name="mystack">
        <interceptor-ref name="timer"></interceptor-ref>
        <interceptor-ref name="logger"></interceptor-ref>
    </interceptor-stack>
</interceptors>
```

定义默认的拦截器，每个 Action 都会自动引用，如果 Action 中引用了其他的拦截器，默认的拦截器将无效。下面引入了“mystack”拦截器栈。

```
<default-interceptor-ref name="mystack"></default-interceptor-ref>
```

4)　配置全局 Results

下面是定义全局 Results 配置。

```
<global-results>
    <result name="input">/error.jsp</result>
</global-results>
```

5)　配置 Action

配置一个 Action 可以被多次映射(只要 Action 配置中的 name 不同)，可以配置如下属性。

- name：Action 名称。
- class：对应的类的路径。
- method：调用 Action 中的方法名。

```
    <action name="hello" class="com.kay.struts2.Action.LoginAction">
```

引用拦截器，可以配置 name 属性，指定要引用的拦截器名称。配置如下。

- name：拦截器名称或拦截器栈名称。

```
    <interceptor-ref name="timer"></interceptor-ref>
```

6) 配置 Result

一个 Result 表示一个可能的输出，它有如下两个配置属性。

- name：Result 名称和 Action 中返回的值相同。
- type：Result 类型不写则选用 superpackage 的 type，struts-default.xml 中的默认为 dispatcher。

```
<result name="success" type="dispatcher">/talk.jsp</result>
```

7) Action 的参数设置

配置 Action 可以将一个请求 URL 映射到一个 action 类。它只有一个 name 属性，该属性对应 Action 中的 get/set 方法。

```
<param name="url">http://www.sina.com</param>
```

3. struts.properties

Struts 2 的属性配置文件，默认叫 default.properties 文件。它配置 Struts 2 的默认配置。这个文件提供了一种更改框架默认行为方式的机制。一般情况下，如果不是打算让调试更加方便的话，无须更改这个文件。



在“struts.properties”文件中定义的属性都可以在“web.xml”文件的“init-param”标签中进行配置，或者通过“struts.xml”文件中的“constant”标签来修改。

default.properties 文件位于 Struts 2 的 jar 包中，因为是只读文件，所以无法修改，如果想要修改 Struts 2 的默认配置该如何做呢？

首先可以在 classpath 的根目录下新建一个 struts.properties 文件。创建了 struts.properties 文件之后，该文件中的属性设置会覆盖 default.properties 文件中的属性设置。

例如：修改 Struts 2 的默认后缀为.do，只需在 struts.properties 文件中写入如下代码即可。

```
struts.action.extension=do
```

为了方便读者查看，下面将 default.properties 文件中的配置常用属性列举出来，如表 2-1 所示。

表 2-1 default.properties 文件常用属性

属 性 名	说 明
struts.action.extension	设置 Struts 2 的后缀，默认为“.action”
struts.configuration	org.apache.struts2.config.Configuration 接口名
struts.configuration.files	Struts 2 自动加载的一个配置文件列表，默认加载 struts-default.xml、struts-plugin.xml、struts.xml
struts.configuration.xml.reload	是否加载 XML 配置，默认为 true

续表

属 性 名	说　明
struts.continuations.package	含有 Action 完整连续的 package 名称
struts.custom.i18n.resources	加载附加的国际化属性文件(不包含.properties 后缀)
struts.custom.properties	加载附加的配置文件的位置
struts.enable.DynamicMethodInvocation	允许动态方法调用，使用通配符动态调用 Action
struts.i18n.encoding	国际化信息内码编号，默认为 UTF-8
struts.i18n.reload	是否国际化信息自动加载
struts.locale	默认的国际化地区信息
struts.multipart.maxSize	multipart 请求信息的最大尺寸(文件上传用)
struts.multipart.parser	专为 multipart 请求信息使用的 org.apache.struts2.dispatcher.multipart.MultiPartRequest 解析器接口(文件上传用)
struts.multipart.saveDir	设置存储上传文件的目录
struts.objectFactory	com.opensymphony.xwork2.ObjectFactory 接口(spring)
struts.objectFactory.spring.autoWire	是否自动绑定 Spring
struts.objectFactory.spring.useClassCache	是否 Spring 应该使用自身的 cache
struts.serve.static.browserCache	是否 Struts 2 过滤器中提供的静态内容应该被浏览器缓存在头部属性中
struts.serve.static	是否 Struts 2 过滤器应该提供静态内容
struts.url.http.port	设置 HTTP 端口
struts.url.includeParams	在 URL 中产生默认的 includeParams
struts.velocity.configfile	velocity 配置文件路径，默认为 velocity.properties
struts.velocity.contexts	velocity 的 context 列表

4．struts-default.xml

struts-default.xml 文件为 Struts 2 框架提供了默认配置，是框架的基础配置文件，这个文件包含在 struts2-core-2.0.11.jar 中，由框架自动加载。

struts-default.xml 文件会自动被包含到 struts.xml 文件中，以提供标准的配置设置而不需要复制其内容。前面章节中在介绍配置 struts.xml 文件时，给出了下面这句代码。

```
<package name="default" extends="struts-default"/>
```

struts-default 包就是在 struts-default.xml 文件中定义的，在这个包中定义了 Struts 2 内置的结果类型(包括 Servlet 转发、Servlet 重定向、FreeMarker 模板输出、XSTL 渲染和 ActionChain Result 等)和内置的拦截器，以及由不同拦截器组成的拦截器栈，这些拦截器栈可以直接使用，也可以作为自定义的拦截器栈的基础。在 struts-default.xml 文件的最后还定义了默认的拦截器引用。

5．struts-plugin.xml

Struts 2 为了扩展自身的功能，提供了类似于 Eclipse 的插件机制，插件以 Jar 包的方式提供。例如集成 Spring 的插件 struts2-spring-plugin-2.0.11.jar，集成 Sitemesh 的插件 struts-sitemesh-plugin-2.0.11.jar，以及集成 Struts 1 的插件 struts2-struts1-plugin-2.0.11.jar 等。

struts-plugin.xml 就是由插件使用的配置文件，该文件的结构和 struts.xml 相同，存放在插件 Jar 包的根路径中。如果读者不是开发插件的话，是不需要编写这个配置文件的，作为普通开发人员，更多的是使用插件。如果读者有兴趣开发插件，可参看 Struts 2 开发包安装目录下的 docs/docs/plugins.html 文档。

下面介绍一个 struts-plugin.xml 文件的例子，用来配置 Spring 插件的配置文件，其代码如下所示。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
<!--定义 Spring 的 ObjectFactory -->
<bean type="com.opensymphony.xwork2.ObjectFactory" name="spring"
 class="org.apache.struts2.spring.StrutsSpringObjectFactory" />
<!--将 SpringObjectFactory 设置为 Struts 2 默认的 ObjectFactory-->
<constant name="struts.objectFactory" value="spring"/>
<!-- 定义 spring-default 包 -->
<package name="spring-default">
<!-- 配置拦截器列表 -->
<interceptors>
<interceptor name="autowiring"
class="com.opensymphony.xwork2.spring.interceptor.
ActionAutowiringInterceptor"/>
<interceptor name="sessionAutowiring"
class="org.apache.struts2.spring.interceptor.
SessionContextAutowiringInterceptor"/>
</interceptors>
</package>
</struts>
```



struts-plugin.xml 文件在插件加载时，被 Struts 2 框架自动读取。

2.1.2　实例描述

通过学习 Struts 2 的基础知识，可能读者对 Struts 2 的配置理解得还不是很透彻，下面将通过一个“小小图书馆”的实例来深入学习和理解 Struts 2 的基本配置。

2.1.3　实例应用

【例 2-1】 小小图书馆。

(1) 新建一个 Struts2_2 项目，目的是做一个小小图书馆。首先导入 Struts 2 相关 jar 包，并在项目 WEB-INF/web.xml 文件中添加 Struts 2 的核心控制器 org.apache.struts2.dispatcher.FilterDispatcher，代码如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app 2 4.xsd">
    <filter>
        <filter-name>struts2</filter-name>

    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-
class>
    </filter>
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern><!-- 配置拦截所有请求 url -->
    </filter-mapping>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

(2) 接着在项目的 src 目录下新建一个 struts.xml 文件，输入如下代码。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 2.0//EN" "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
</struts>
```

(3) 在 Struts2_2 项目目录下，新建一个 login.jsp 文件，代码如下所示。

```
<%@ page contentType="text/html; charset=utf-8" language="java"
import="java.sql.*" errorPage="" %>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Login</title>
</head>
```

```
<body>
    <s:form action="GetBooks.action">
        <s:textfield name="username" label="用户名" />
        <s:password name="password" label="密码" />
        <s:submit value="提交" />
    </s:form>
</body>
</html>
```

(4) 在 src/books 目录下建立 Action 接口，代码如下所示。

```
package books;
public interface Action
{
    public static String SUCCESS = "success";
    public static String NONE = "none";
    public static String ERROR = "error";
    public static String INPUT = "input";
    public static String LOGIN = "login";

    public String execute() throws Exception;
}
```

(5) 定义一个储存数据的 BooksService.java(实际中不可能这样用，这里只是为了方便演示)，代码如下所示。

```
package books;
public class BooksService
{
    //下面是要用的图书数据，在此为了方便演示，所以没有从数据库中加载数据。
    private String[] books = new String[]{"Spring2.0 宝典","轻量级 J2EE 企业应用
实战","基于 Ajax 的在线订购系统","Struts,spring ,habernate 整合开发"};
    //获取图书信息数据方法，此方法相当于连接数据时，从数据库中查询获取图书信息数据。
    public String[] getLeeBooks()
    {
        return books;
    }
}
```

(6) 图书馆是要登录的，既然要登录，那么登录验证是必不可少的，登录进去之后可以看到馆内的图书信息列表，所以需要定义一个登录验证并转入书籍列表类 GetBooksAction.java，代码如下所示。

```
package books;
import com.opensymphony.xwork2.ActionContext;
public class GetBooksAction implements Action
{
    private String[] books;     //图书信息数据
    private String username;    //用户名
    private String password;    //密码
```

```
public void setBooks(String[] books)    //设置图书信息数据
{
    this.books = books;
}

public String[] getBooks()          //获取图书信息数据
{
    return books;
}

public String getUsername()         //获取用户名
{
    return this.username;
}

public String getPassword()         //获取密码
{
    return this.password;
}

public void setUsername(String username)    //设置用户名
{
    this.username = username;
}
public void setPassword(String password)    //设置密码
{
    this.password = password;
}

public String execute()throws Exception
{
    //验证用户登录的用户名与密码是否正确
    if (getUsername().equals("admin")&&getPassword().equals("admin"))
    {
        //用户名正确将用户名放入到 ActionContext 的 Session 中 “user” 参数
        ActionContext.getContext().getSession().put("user",getUsername());
    }
    else
    {
        //用户名不正确将 “error” 放入到 ActionContext 的 Session 中 “user” 参数
        ActionContext.getContext().getSession().put("user","error");
    }
    //从 Session 中取出 “user” 参数
    String user = (String)ActionContext.getContext().getSession(). get
    ("user");
    System.out.println((String)ActionContext.getContext().getSession().
    get ("user"));
    //判断 “user” 是否 equalse("admin")
```

```
        if(user!=null&&user.equals("admin"))
        {
            //登录成功了，创建 BooksService 业务类，调用 bs 的 getLeeBooks()方法获取图
              书信息
            BooksService bs = new BooksService();
            setBooks(bs.getLeeBooks());      //设置 Action 的 books
            return SUCCESS;                  //返回登录成功
        }
        else
        {            //否则返回重新登录
            return LOGIN;
        }
    }
}
```

(7) GetBooksAction 创建好之后，需要在 struts.xml 文件中配置 Action，代码如下所示。

```
<include file="struts-default.xml"></include>
<package name="Struts2 2" extends="struts-default">
    <action name="GetBooks" class="books.GetBooksAction">
        <result name="login">/login.jsp</result>
        <result name="success">/showBooks.jsp</result>
    </action>
</package>
```

(8) 最后一步需要创建一个显示书籍页面 showBooks.jsp，代码如下所示。

```
<%@ page contentType="text/html; charset=utf-8" language="java"
import="java.sql.*" errorPage="" %>
<%@ page import="java.util.*,com.opensymphony.xwork2.util.*"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>showBooks</title>
</head>
<body>
    <s:iterator value="books" status="index">  <!—使用 struts2 迭代标签,迭代
books 数组 -->
    <s:if test="#index.odd==true"> <!-- 使用 struts2 判断标签 -->
        <span style="color:#FF0000"><s:property /></span><br />
    </s:if>
    <s:else>
        <s:property /><br />
    </s:else>
    </s:iterator>
</body>
</html>
```

2.1.4　运行结果

现在来看看所建立的“小小图书馆”。运行的 login.jsp 页面。执行效果如图 2-1 所示。



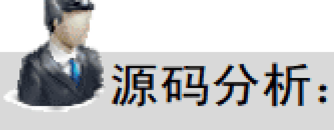
图 2-1　图书馆登录界面

当用户输入用户名 admin 和密码 admin，单击【登录】按钮时，如果登录成功，读者可以看到图书馆的图书信息，如果登录失败，将让用户再次登录。执行显示图书信息的效果如图 2-2 所示。



图 2-2　图书信息

2.1.5　实例分析


源码分析：

上述实例创建一个 login.jsp 页面，用于用户登录该图书馆系统，并通过创建 GetBooksAction 来验证用户登录，登录成功之后，创建 BooksService 来载入显示图书信息。最终在 showBooks.jsp 页面中显示图书信息。

2.2 配置 Struts 2 的命名空间

视频教学：光盘/videos/02/bean.avi 长度：9 分钟
光盘/videos/02/constant.avi 长度：7 分钟
光盘/videos/02/ package.avi 长度：5 分钟
光盘/videos/02/ namespace.avi 长度：9 分钟
光盘/videos/02/ Include.avi 长度：5 分钟

2.2.1 基础知识——深入 Struts 2 的配置文件

Struts 2 的 struts.xml 配置文件的结构包括该文件的根元素及每个元素能包含的子元素等。本节将讲解深入配置，包括 Bean 的配置、常量配置、包配置和命名空间配置等。

1. Bean

Struts 2 框架以可配置的方式来管理 Struts 2 的核心组件，从而允许开发者可以很方便地扩展该框架的核心组件。当开发者需要扩展或者替换 Struts 2 的核心组件时，只需要提供自己的组件类，并将该组件实现类部署在 Struts 2 的 IOC 容器即可。

在 struts.xml 文件中定义 Bean 时，通常有如下两个作用。

(1) 框架的 IOC 容器创建 Bean 的实例，然后将该实例注入框架的内部对象中。

(2) 通过 Bean 的静态方法向 Bean 注入值。

在第一种用法中，Bean 将被注入框架内部，和内部对象协作，框架要知道 Bean 的类型，因此在配置 Bean 时，通常要使用 type 属性，以指明 Bean 实现的接口。如创建一个 ObjectFactory，需要在 struts.xml 文件中使用 Bean 元素配置，代码如下所示。

```
<bean type="com.opensymphony.xwork2.ObjectFactory" name="myfactory"
class="com.company.myapp.MyObjectFactory"/>
```

对于第二种用法，使用值注入，允许不创建 Bean，而是让 Bean 接收框架的常量。Bean 使用值注入，必须使用 static 属性，并将该属性设置为 true。例如配置 FilterDispatcher。

```
<bean class="org.apache.struts2.dispatcher.FilterDispatcher" static="true"/>
```



对于绝大部分 Struts 2 应用而言，因为无需重新定义 Struts 2 框架的核心组件，因此也就无需在文件中定义 Bean。

Bean 元素的几个属性如表 2-2 所示。

2. 常量

通过配置常量，可以改变 Struts 2 框架和插件的行为，从而满足不同 Web 应用的需求。实际上，配置常量就是配置 Struts 2 的属性。常量可以在多个文件中声明。默认地，Struts 2 框架按照下列文件的顺序搜索常量，越靠后的文件优先级越高，也就是说，顺序靠后的文件中的常

量设置可以覆盖顺序靠前的文件中的常量设置。文件加载顺序如下所示。

(1) struts-default.xml：存放在 struts2-core-x.x.x.jar 文件中。

(2) struts-plugin.xml：存放在 struts2-xxx-x.x.x.jar 等 Struts 2 插件的 Jar 文件中。

(3) struts.xml：Web 应用中默认的 Struts 2 配置文件。

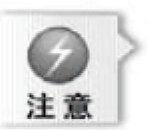
(4) struts.properties：Struts 2 的属性配置文件。

(5) web.xml：Web 应用的配置文件。

表 2-2　Bean 元素的属性

属　性	是否必需	说　明
class	是	bean 的类名
type	否	bean 类实现的接口
name	否	bean 的名字，在具有相同 type 属性的 bean 中，该名字必须是唯一的
scope	否	bean 的范围，有效的值包括：default、singleton、request、session 和 thread
static	否	是否使用静态方式注入，如果指定了 type 属性，则不要把该属性设为 true
optional	否	bean 是否是可选的

在 struts.xml 文件中，通过<constant>节点配置常量(Constant)，可以作为指定 Struts 2 属性的一种方式。而 Struts 2 的常量既可以在 struts.xml 文件中配置，也可以在 struts.properties 文件中配置，在其他一些配置文件中也可以实现。



如果在多个文件中配置同一个 Struts 2 常量，按照加载顺序，后一个文件中配置的常量值将会覆盖前面文件中配置的常量值。

使用<constant>元素配置常量时，需要指定以下两个必填属性。

- name：指定常量的名称。
- value：指定常量的属性值。

例如，在 struts.xml 文件中，指定字符编码集为 gb2312，代码如下。

```
<constant name="struts.i18n.encoding" value="gb2312"/>
```

在 struts.xml 文件中，指定国际化资源文件的 basename 为 globalMessages，代码如下。

```
<constant name="struts.custom.i18n.resources" value="globalMessages" />
```

如果使用 struts.properties 文件实现上述常量的配置，代码如下。

```
struts.i18n.encoding=gb2312
struts.custom.i18n.resources=globalMessages
```



在 struts.properties 文件中，文件的内容使用键值对 Key-Value 的形式，其中 Key 对应 Struts 2 常量中的 name，Value 对应 Struts 2 常量的 value 等。

使用 web.xml 文件同样可以实现上述常量的配置，代码如下所示。

```
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>
        org.apache.struts2.dispatcher.FilterDispatcher
```

```
    </filter-class>
    <init-param>
        <param-name>
            struts2.custom.i18n.resources
        </param-name>
        <param-value>globalMessages</param-value>
    </init-param>
    <init-param>
        <param-name>struts.i18n.encoding</param-name>
        <param-value>gb2312</param-value>
    </init-param>
</filter>
```

通过上述代码可以看出，在 web.xml 文件中配置常量时，需要在<filter>元素中使用子元素<init-param>来实现。



如果将 Struts 2 常量配置在 web.xml 文件中，实现同样的功能，代码量会明显增加，而且降低了 web.xml 文件的可读性。所以推荐将 Struts 2 常量配置在 struts.xml 文件进行集中管理。

3. 包

在 Struts 2 框架中，其核心组件是 Action 和拦截器等，该框架使用包来管理这些组件。在包中可以配置多个 Action、多个拦截器或者是多个拦截器引用的集合等。使用<package>元素配置包时，可以指定 4 个属性，如表 2-3 所示。

表 2-3 <package>元素的属性

属 性 名	必 选	说 明
name	是	指定包的名称，该名称是该包被其他包引用的 key 值
extends	否	指定该包继承的其他包
namespace	否	指定该包的命名空间
abstract	否	指定该包是否是一个抽象包。抽象包中不能定义 Action



如果使用 extends 继承其他包，则子包可以继承父包中的拦截器和 Action 等。但是父包必须在子包前面定义。

例如，在 struts.xml 文件中配置两个包，配置方式的代码如下所示。

```
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name="struts.i18n.encoding" value="gb2312"/>
    <package name="default" extends="struts-default">  <!--配置 default 包-->
        <interceptors>
            <interceptor-stack name="myStack">
                <interceptor-ref name="defaultStack"/>
```

```
                <interceptor-ref name="myInterceptor"/>
            </interceptor-stack>
        </interceptors>
        <action name="action1" class="ActionClass1">       <!--配置 Action-->
            <result name="success">/success.jsp</result>
        </action>
    </package>
    <package name="myPackage" extends="default" namespace="/skill">
                                    <!--配置 myPackage 包，并且指定命名空间-->
        <default-interceptor-ref name="myStack"/>      <!--myStack 拦截器-->
        <action name="action2" class="ActionClass2">   <!--配置 Action-->
            <result name="success">/success.jsp</result> <!--配置 Result-->
            <interceptor-ref name="defaultStack"/>     <!--使用默认拦截器-->
            <interceptor-ref name="myInterceptor"/>    <!--使用 myStack-->
        </action>
    </package>
</struts>
```

上述文件中，配置 default 包，配置 myPackage 包，在包中配置有拦截器和 Action。其中，default 包继承 Struts 2 框架的默认包 struts-default；myPackage 包继承 default 包。

4．命名空间

Struts 2 通过为包指定 namespace 属性来为包下面的所有 Action 指定共同的命名空间，同一个命名空间不能有同名的 Action。

```
<package name="struts2" extends="struts-default">
<package name="my" extends="struts-default" namespace="/manage">
```

上面配置了两个包：struts2 和 my，配置 my 包时指定了该包的命名空间为/manage。

- 包 struts2：没有指定 namespace 属性。如果某个包没有指定 namespace 属性，即该包使用默认的命名空间，默认的命名空间总是""。
- 包 my：指定了命名空间/manage，因此该包下所有的 Action 处理的 URL 应该是“命名空间/Action 名”。

Struts 2 先在指定的路径下找 Action，如果找不到则会去默认的路径下找 Action。

5．包含(Include)配置

在大型的 Web 项目中，为了降低项目的复杂度，便于团队成员分工合作，通常将项目划分为多个小模块，每个模块单独开发与管理。Struts 2 也提供了这种“分而治之”的策略。我们可以为每个小模块提供一个配置文件，对其进行配置，然后在 struts.xml 文件中使用 include 元素来包含其他的配置文件。

include 元素必须有一个属性 file，指定被包含文件的文件名。例如使用 include 元素，在 struts.xml 文件中引入一个 struts-tags.xml 文件，代码如下所示。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
```

```
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <include file="struts-tags.xml"/>
</struts>
```

2.2.2 实例描述

在网上论坛中有人提出了一个问题，说他定义了一个 LoginAction 用于登录系统，在 struts.xml 文件中也配置好了，但当他使用“项目名/login.action”时，没有访问到这个 Action，这是怎么回事呢？

这个问题深深地吸引了我，我让他把他的 LoginAction 和 struts.xml 文件中的配置代码，粘贴到论坛中，我帮他看看，我一看，恍然大悟呀，他使用了命名空间的配置。知道他的出错原因了，这就好办了，将请求 URL 中添加入命名空间，即可请求成功。本节的实例将演示 Struts 2 命名空间的配置以及如何请求使用。

2.2.3 实例应用

【例 2-2】 配置 Struts 2 的命名空间。

(1) 首先在项目的 src 目录下新建一个 com.namespace 包，在该包下新建一个 NameSpace Action，用来处理用户请求，代码如下所示。

```
public class NameSpaceAction {
    private String username;        //用户名
    private String password;        //密码

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String execute(){
        System.out.println("execute方法被调用");
        return "SUCCESS";
    }
```

```
    public String methodTest(){
        System.out.println("methodTest 方法被调用 aaaaaaaa");
        return "SUCCESS";
    }
}
```

(2) 在 src 目录下的 struts.xml 文件中配置 NameSpaceAction，并指定不同的 namespace 来配置 Action，代码如下所示。

```
<!-- 默认命名空间,即不写 namespace 或者写 namespace="",在前端不指明命名空间
的时候或随便写非命名空间名字的时候用到-->
<package name="struts2" extends="struts-default">
        <action name="login" class="com.namespace.NameSpaceAction">
            <result name="SUCCESS">/jsp/success.jsp</result>
        </action>
    </package>
    <package name="test1" extends="struts-default" namespace="/test1">
        <action name="login" class="com.namespace.NameSpaceAction">
            <result name="SUCCESS">/jsp/test1.jsp</result>
        </action>
    </package>

    <package name="test2" extends="struts-default" namespace="/test2">
        <action name="login" class="com.namespace.NameSpaceAction">
            <result name="SUCCESS">/jsp/test2.jsp</result>
        </action>
    </package>
    <package name="test3" extends="struts-default" namespace="/test3">
        <action name="login" class="com.namespace.NameSpaceAction">
            <result name="SUCCESS">/jsp/test3.jsp</result>
        </action>
    </package>
    <!-- 根命名空间，只用/来命名命名空间的时候用到 -->
    <package name="test4" extends="struts-default" namespace="/">
        <action name="login" class="com.namespace.NameSpaceAction">
            <result name="SUCCESS">/jsp/test4.jsp</result>
        </action>
    </package>
    <!-- 执行非 execute 方法的配置方式-->
    <package name="struts22" extends="struts-default" namespace="/test5">
        <action name="login1" class="com.namespace.NameSpaceAction">
            <result name="SUCCESS">/jsp/test5.jsp</result>
        </action>
        <action name="login2" class="com.namespace.NameSpaceAction"
method="methodTest">
            <result name="SUCCESS">/jsp/test5.jsp</result>
        </action>
    </package>
```

(3) 在项目下新建一个 namespace.jsp 页面，在这个页面中定义一个登录表单和几个链接，分别使用不同的命名空间发出请求，代码如下所示。

```
<form action="<%=basePath%>login.action" method="post">
用户名:<input type="text" name="username"><br>
密码:<input type="text" name="password"><br>
<input type="submit" value="登录" >
</form>
<br>
<a href="<%=basePath%>test1/login.action">命名空间 1 进行登录</a>
<a href="<%=basePath%>test2/login.action">命名空间 2 进行登录</a>
<a href="<%=basePath%>test3/login.action">命名空间 3 进行登录</a>
<a href="<%=basePath%>/login.action">根命名空间进行登录</a><!-- 根命名空间 -->
<a href="<%=basePath%>test5/login1!methodTest.action">命名空间 51 进行登录</a>
<a href="<%=basePath%>test5/login2.action">命名空间 52 进行登录</a>
```

(4) 使用默认命名空间登录成功后跳转到登录成功提示页面，代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>默认命名空间</title>
    </head>
    <body>
        默认命名空间,登录成功! <br>
    </body>
</html>
```

(5) 由于上面 namespace.jsp 页面中的各个链接单击发出请求不同命名空间的 login.action，最终跳转到 Action 配置成功的页面如 test1 命名空间，即跳转到/jsp/test1.jsp 页面，该页面代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>test1 命名空间</title>
    </head>
    <body>
        test1 命名空间,登录成功! <br>
    </body>
</html>
```



本实例还有 test2.jsp、test3.jsp、test4.jsp、test5.jsp，这些和 test1.jsp 差不多一样，只是提示请求某个命名空间的 login.action 登录成功了。在此不再作介绍，具体可查看 Struts2_2/jsp 目录下这几个文件的源码。

2.2.4 运行结果

运行 namespace.jsp 页面，单击链接，请求不同命名空间的 Action，进行登录，执行效果

如图 2-3 所示。单击下面各个命名空间登录，如：单击“命名空间 1 进行登录”，执行效果如图 2-4 所示。



图 2-3　不同的命名空间登录



图 2-4　test1 命名空间登录成功

2.2.5　实例分析



源码分析：

在本实例中，我们将 NameSpaceAction 在 struts.xml 文件中配置到不同的命名空间中，这样便可以在请求 URL 地址中添加入不同的命名空间，如“http://localhost:8080/Struts2_2/test1/login.action”。至此，相信读者已经知道如何配置使用 struts.xml 文件中的命名空间了。

2.3　管 理 用 户

Action 的配置是 Struts 2 框架的一个基础工作单元，每一个 Action 的配置都有对应的处理类，当一个请求和 Action 的 name 相匹配时，框架将根据所配置的 Action 映射来决定对请求的处理。Struts 2 Action 几乎完全吸收了 WebWork 的精华，本节将先总结一下 Action 的配置方法。

 视频教学：光盘/videos/02/action.avi　　　　长度：8 分钟

2.3.1　基础知识——Action 的配置

首先来看一下 com.opensymphony.xwork2.Action 的接口声明，Action 提供 execute()方法，子类必须实现 execute()方法。

```
public interface Action {
    public String execute() throws Exception;
}
```



com.opensymphony.xwork2.ActionSupport 是 com.opensymphony.xwork2.Action 的默认实现，实现了 execute()方法。
Action 通常继承 com.opensymphony.xwork2.ActionSupport。

1．简单的 Action 配置

如下所示为一个 Action 的配置代码。

```
<action name="logon" class="tutorial.Logon">
    <result type="redirect-action">Menu</result>
    <result name="input">/tutorial/Logon.jsp</result>
</action>
```

前台发送调用 logon.action 的请求，监听器调用默认的 execute()方法。

2．一个方法一个 Action 配置

在开发中通常将多个方法写在一个 Action 中，这样 Action 便可以采用集中配置方式。下面来看看如何用一个方法配置一个 Action，代码如下所示。

```
<action name="delete" class="example.CrudAction" method="delete">
```

3．通配符方式

采用“*”号通配符来配置 Action，代码如下所示。

```
<action name="*Crud" class="example.Crud" method="{1}">
```

method="{1}"表示第一个通配符是方法。
Action 调用的例子：addCrud.action、deleteCrud.action、updateCrud.action、viewCrud.action。

4．分割符加上*

分割符可以是“_”、“！”等。下面使用“_”分割符来配置 Action，代码如下所示。

```
<action name="crud_*" class="example.Crud" method="{1}">
```

调用方式：crud_add.action、crud_delete.action。
“!”的使用是否和“_”一样呢？它们是不一样的，“!”代码如下所示。

```
<action name="crud!*" class="example.Crud" method="{1}">
```

调用方式：crud!add.action、crud!delete.action。

5．URL 映射规则

“/User/add.action”调用 User 的 add 方法，其配置代码如下所示。

```
<action name="*/*" method="{2}" class="com.infoq.actions.{1}Action">
    <result type="redirect">/{1}/view.action</result>
    <result name="view">/{1}/view.jsp</result>
    <result name="input">/{1}/edit.jsp</result>
    <result name="home">/{1}/home.jsp</result>
</action>
```

2.3.2　实例描述

昨天夜里突然接到一个朋友的电话，当时他在加班，在看公司以前的一个项目，这个项目现在要进行升级优化。但这个项目的 Struts 2 配置文件写的太乱了，而且代码很多。他看了很久也不知道如何着手修改，万分着急和无奈之下，他向我抱怨工作难度太大。的确实如此，配置文件多本来就是件让人头疼的事，对他我深表同情。所以，读者将来编程时，一定不要使用太多的配置文件，可以采用通配符、分割符等，将一些类似的 Action 配置统一配置成一个 Action，这样别人看起来也有条理性，配置文件也会减少许多。别人看你的程序也就容易多了。

本节的实例是如何只在 struts.xml 文件中配置一个 Action，就可以对用户进行“增删查改”的操作。

2.3.3　实例应用

【例 2-3】 管理用户。

(1) 实际开发中，对用户的管理操作是非常频繁的，现在做一个管理用户的实例，主要就是“增”、“删”、“查”、“改”。在项目的 src 下创建一个 com.cqxs.action 包，在该包下新建一个 UserAction，用于处理用户操作的请求，代码如下所示。

```
package com.cqxs.action;
import com.opensymphony.xwork2.ActionSupport;
public class UserAction extends ActionSupport{

    public String add(){                //添加用户
        return SUCCESS;
    }

    public String delete(){             //删除用户
        return SUCCESS;
    }

    public String update(){             //修改用户
        return SUCCESS;
    }

    public String find(){               //查找用户
        return SUCCESS;
    }
}
```

(2) 在 struts.xml 文件中添加 UserAction 的配置，总共有三种配置方案，不过，本案例中使用的是第二种方案，代码如下所示。

```
<package name="HelloWord" extends="struts-default">
    <!-- 第一种：通常的配置方式
    <action name="user" class="com.cqxs.action.UserAction" method="add">
        <result>/User add success.jsp</result>
    </action>-->
```

```
    <!-- 第二种：简化配置 -->
    <action name="User *" class="com.cqxs.action.UserAction" method="{1}">
        <result>/User {1} success.jsp</result>
    </action>
        <!--   第三种：使用通配符 最大限度的简化配置
    <action name="* *" class="com.cqxs.action.{1}Action">
        <result>/{1} {2} success.jsp</result>
    </action>   -->
</package>
```

(3) 在项目根目录下新建一个 user.jsp 页面，在这个页面中显示用户的信息，并可以通过单击链接执行“增”、“删”、“查”、“改”操作，代码如下所示。

```
<table align="center" style="color:#000000">
    <thead><tr ><td>用户名</td><td>密码</td><td>年龄</td><td>性别
</td></tr></thead>
    <tbody>
        <tr><td>sky</td><td>123456</td><td>23</td><td>女</td></tr>
        <tr><td>blue</td><td>1bule</td><td>28</td><td>男</td></tr>
        <tr><td>雨后彩虹</td><td>type</td><td>45</td><td>男</td></tr>
        <tr><td>小河</td><td>123</td><td>15</td><td>女</td></tr>
        <tr>
            <td><a href="User add.action">添加用户・</a></td>
            <td><a href="User delete.action">删除用户</a></td>
            <td><a href="User update.action">修改用户</a></td>
            <td><a href="User find.action">查询用户</a></td>
        </tr>
    </tbody>
</table>
```

(4) 当单击“添加用户”链接发出“User_add.action”请求成功时，将跳转到添加用户成功页面 User_add_success.jsp，代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head></head>
    <body>
        用户：添加成功<br>
    </body>
</html>
```

(5) “删除用户”成功页面 User_delete_success.jsp 的代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <title>删除用户成功</title>
    </head>
    <body>
        用户：删除成功<br>
```

```
    </body>
</html>
```

(6)　“查找用户”成功页面 User_find_success.jsp 的代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>My JSP 'success.jsp' starting page</title>
    </head>
    <body>
        用户：查找成功<br>
    </body>
</html>
```

(7)　“修改用户”成功页面 User_update_success.jsp 的代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>My JSP 'success.jsp' starting page</title>
    </head>
    <body>
        用户：修改成功<br>
    </body>
</html>
```

2.3.4　运行结果

用户管理实例做好后，其执行效果如图 2-5 所示。



图 2-5　管理用户

2.3.5　实例分析



源码分析：

上述案例中对用户进行了“增、删、查、改”4 个操作，这 4 个操作也就对应了 4 个请求。以前编写 Action 时，一个请求操作对应一个 Action。但是这样会导致 struts.xml 文件中配置的 Action 急剧增多，不便于管理，而且后期维护修改工作量也会大量增多。

因此本案例采用了 Action 的通配符配置，定义了一个 UserAction，将对用户的一切操作定义成不同的方法，如：add、delete、update、select 方法。同时在 struts.xml 文件中使用通配符配置一个 Action，就可以对应处理多个请求，这样 Action 也轻巧多了。

2.4　部门信息管理

Action 的配置很重要，不过 Result 配置也是相当重要的。没有 Result 配置，程序就没有执行结果，Result 配置不当或错误，程序也会执行出错的结果。所以对于 Result 的配置读者要认真学习，以达到理解并灵活应用的程度。



视频教学：光盘/videos/02/result.avi　　　　长度：7 分钟

2.4.1　基础知识——Result 配置

在前面的许多案例中，所用到的 Action 基本都继承自 ActionSupport 这个类。这个类中定义了五个字段：SUCCESS，NONE，ERROR，INPUT，LOGING。我们可以直接返回这些字段值，这些字段值的实质是被定义成：String SUCCESS="success"的形式，所以只要在 Result 元素中用它们的小写即可。

<result>标准完整形式如下。

```
<result name="success" type="dispatcher">
        <param name="location">/default.jsp</param>
</result>
```

如果都采用默认的形式，最终可以简写成如下形式。

```
<result>/default.jsp</result>
```

下面是 Result 类型列表，共同认识一下，如表 2-4 所示。

表 2-4　Result 类型

type 类型值	作用说明	对 应 类
chain	用来处理 Action 链	com.opensymphony.xwork2.ActionChainResult
dispatcher	用来转向页面，通常处理 JSP	org.apache.struts2.dispatcher.ServletDispatcherResult
redirect	重定向到一个 URL	org.apache.struts2.dispatcher.ServletRedirectResult
redirectAction	重定向到一个 Action	org.apache.struts2.dispatcher.ServletActionRedirectResult
plainText	显示源文件内容，如文件源码	org.apache.struts2.dispatcher.PlainTextResult
freemarker	处理 freemarker 模板	org.apache.struts2.views.freemarker.FreemarkerResult
httpheader	控制特殊 http 行为的结果类型	Org.apache.struts2.dispatcher.HttpHeaderResult
stream	向浏览器发送 InputStream 对象，通常用来处理文件下载，还可用于返回 Ajax 数据	org.apache.struts2.dispatcher.StreamResult
velocity	处理 Velocity 模板	org.apache.struts2.dispatcher.VelocityResult
xslt	处理 XML/XLST 模板	org.apache.struts2.views.xslt.XSLTResult

Result 类型很多，不过常用的也没几个，在此将挑选几个常用的类型进行详细讲解。

1．dispatcher

dispatcher 为默认的 result 类型，一般情况下在 struts.xml 中会写出如下代码。

```
<result name="success">/main.jsp</result>
```

以上写法使用了两个默认，其完整的写法如下所示。

```
<result name="success" type="dispatcher">
        <param name="location">/maini.jsp</param>
</result>
```

第一个默认：type="dispatcher"；第二个默认：设置的为 location 参数，location 只能是页面，不能是另一个 Action(可用 type="chain"解决)。

2．redirect

redirect 可以重定向到一个页面，另一个 Action 或一个网址。

```
<result name="success" type="redirect">aaa.jsp</result>
<result name="success" type="redirect">bbb.action</result>
<result name="success" type="redirect">www.baidu.com</result>
```

3．chain

chain 主要用于把相关的几个 Action 连接起来，共同完成一个功能。

```
<action name="step1" class="test.Step1Action">
    <result name="success" type="chain">step2.action</result>
</action>
<action name="step2" class="test.Step2Action">
```

```
    <result name="success">finish.jsp</result>
</action>
```

查看 execute()方法，主要思想如下。

```
// 根据 Action 名称 finalActionName 及要调用的方法 finalMethodName 来 new 一个代理对象
proxy，并执行之
proxy = actionProxyFactory.createActionProxy(finalNamespace, finalActionName,
finalMethodName,
extraContext);
proxy.execute();
```

多个 Action 间数据的传递，主要有两种方式。

- 由于处于 chain 中的 Action 属于同一个 HTTP 请求，共享一个 ActionContext，故可以在上下文中获取，在页面上可以直接使用。手动获取的方法如下。

```
HttpServletRequest request = ServletActionContext.getRequest();
String s=(String)request.getAttribute("propName");
```

- 实现 ModelDriven 接口，在 Step1Action 中加入 getModel，代码如下所示。

```
public Object getModel() {
    return message;
}
```

在 Step2Action 中加入 setModel，代码如下所示。

```
public void setModel(Object o){
    System.out.println("message is:"+o);
}
```



setModel 的调用先于 execute()方法后于构造方法。

2.4.2 实例描述

前天有个新来的同事问了我一个问题——Result 只能配置跳转到 JSP 页面吗？答案当然是“否定”的。只是在 Result 的配置默认情况下是指定跳转到 JSP 页面，因此我们也可以配置重定向到一个 URL 或重定向到一个 Action 等。下面将通过实例来看看如何配置重定向到一个 Action。

2.4.3 实例应用

【例 2-4】 部门信息管理。

(1) 新建一个 Struts 2 App 项目，在该项目的 src 目录下新建一个 struts.org.db 包，然后在该包下新建一个 Dept.java，最后封装一个部门实体类，代码如下所示。

```
public class Dept implements Serializable {
    private int deptId;          //部门 Id
    private String deptName;         //部门名称
    private int deptNum;             //部门编号
    private String deptDesc;         //部门简介
    public Dept() {              //构造函数
        // TODO Auto-generated constructor stub
    }

    public Dept(int deptId, String deptName, int deptNum, String deptDesc) {
    //带参构造
        super();
        this.deptId = deptId;
        this.deptName = deptName;
        this.deptNum = deptNum;
        this.deptDesc = deptDesc;
    }

    public int getDeptId() {
        return deptId;
    }
    public void setDeptId(int deptId) {
        this.deptId = deptId;
    }
    //下面是 deptNum、deptName 和 deptDesc 三个属性的 get、set 方法，在此省略
}
```

(2) 对部门信息进行添加、删除、查看和修改 4 个操作。因为需要连接数据库，所以需要封装一个操作数据库类 Test.java，代码如下所示。

```
public class Test {
    private String username="root";     //数据库用户名
    private String password="123456";   //数据库密码
    private String
url="jdbc:mysql://localhost:3309/dept?useUnicode=true&characterEncoding=utf
8";
    private String driver="com.mysql.jdbc.Driver";     //mysql 驱动类
    public Test() {

    }
    public Connection getConnection()throws Exception{
        Class.forName(driver);          //加载 mysql 驱动
       Connection conn = DriverManager.getConnection(url, username,password);
        //获取数据库连接对象 conn
        return conn;                    //返回连接对象 conn
    }
    public Statement getStatement(){
        Statement st = null;
        //声明一个 Statement 对象，用于执行静态 SQL 语句并返回它所生成结果的对象
```

```
        try {
            st = getConnection().createStatement();    //获取 Statement 对象实例
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return st;              //返回 Statement 对象 st
    }
    public List getDepts(){     //获取部门信息列表
        List<Dept> list = new ArrayList<Dept>();
                                //创建一个存储部门信息的列表对象 list

        try{
            String sql="select * from dept";          //查询 sql 语句
            ResultSet rs=getStatement().executeQuery(sql);
                                //执行查询 sql 语句，返回结果集 rs
            while(rs.next()){
                Dept dept = new Dept();               //创建一个部门对象
                int deptid = rs.getInt("deptid");     //获取结果集中 deptid 的值
                String deptName=rs.getString("deptname");
                int deptnum = rs.getInt("deptnum");
                String deptDesc =rs.getString("deptdesc");
                dept.setDeptId(deptid);
                                //将从结果集中获取的 deptid 赋给部门对象的 deptid 属性
                dept.setdeptName(deptName);
                dept.setDeptNum(deptnum);
                dept.setDeptDesc(deptDesc);
                list.add(dept); //将 dept 添加到 list 列表中
            }
            rs.close();         //关闭 rs
            getStatement().close();            //关闭 st
            getConnection().close();           //关闭 conn
        }catch(Exception e){
            e.printStackTrace();
        }
        return list;                           //返回查询到的部门信息列表对象
    }
    public int getMaxId(){                     //获取最大的部门编号
        int max=0;
        try{
            String sql="select max(deptid) from dept";
            ResultSet rs=getStatement().executeQuery(sql); //查找最大的部门编号
            while(rs.next()){
                int deptid=rs.getInt(1);       //获取最大的部门编号
                max=deptid+1;                  //最大的部门编号再加 1
            }
            rs.close();
            getStatement().close();
```

```
            getConnection().close();
        }catch(Exception e){
            e.printStackTrace();
        }
        return max;     //返回 max 值
    }
    public void insertDept(String deptName,int deptNum,String deptDesc){
                        //添加一个新部门
        int deptId=getMaxId();
                        //将目前最大的部门编号再加 1，就得到了添加新部门的 deptId
        try{
            String sql="insert into dept(deptId,deptName,deptNum,deptDesc)
            values("+deptId+",'"+deptName+"',"+deptNum+",'"+deptDesc+"')";
                        //添加了一个新部门的信息语句
            getStatement().executeUpdate(sql); //执行添加新部门的 sql 语句
            getStatement().close();
            getConnection().close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
    public Dept prepareUpdateDept(int deptId){ //通过 deptId 查找部门
        Dept dept = new Dept();
        try{
            String sql="select * from dept where deptid="+deptId;
                                    //查找 deptId 部门信息的 sql 语句
            ResultSet rs=getStatement().executeQuery(sql); //执行查找 sql 语句
            while(rs.next()){
                dept.setDeptId(rs.getInt("deptid"));
                            //将结果集中 deptid 的值赋给 dept 的 deptId 属性
                dept.setdeptName(rs.getString("deptname"));
                dept.setDeptNum(rs.getInt("deptnum"));
                dept.setDeptDesc(rs.getString("deptdesc"));
            }
            rs.close();
            getStatement().close();
            getConnection().close();
        }catch(Exception e){
            e.printStackTrace();
        }
        return dept;          //返回查找到的部门对象
    }
    public void updateDept(int deptId,String deptName,int deptNum,String
deptDesc){
        try{
            String sql="update dept set
deptname='"+deptName+"',deptnum="+deptNum+",deptdesc='"+deptDesc+"' where
deptid="+deptId;
```

```
            getStatement().executeUpdate(sql); //执行 sql
            getStatement().close();
            getConnection().close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
    public void deleteDept(int deptId){         //删除编号为 deptId 的部门信息记录
        try{
            String sql="delete from dept where deptid="+deptId;
                                //删除编号为 deptId 部门信息记录的 sql 语句
            getStatement().execute(sql);        //执行 sql 语句
            getStatement().close();
            getConnection().close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

(3) 封装定义 DeptAction，这个 Action 包含对应的“添加”、“删除”、“修改”和“查看”四个操作的请求方法，代码如下所示。

```
public class DeptAction extends ActionSupport {

    public DeptAction() {
        // TODO Auto-generated constructor stub
    }
        public String selectDept() {    //查询部门信息
        HttpServletRequest request=ServletActionContext.getRequest();
        Test test = new Test();     //创建一个 Test 类对象
        List depts=test.getDepts();
                            //调用 getDepts()方法返回得到部门信息列表
                              depts
        request.setAttribute("depts",depts);    //将该列表对象存入 request 对象中
        return "success";                       //返回请求成功
    }
    public String insertDept() throws UnsupportedEncodingException{
                                            //添加部门
        Test test = new Test();     创建一个 Test 类对象
        HttpServletRequest request=ServletActionContext.getRequest();
    //request 对象
        String deptName=request.getParameter("deptName");      //获取部门名称
        int deptNum = Integer.valueOf(request.getParameter("deptNum"));
    //获取部门编号
        String deptDesc = request.getParameter("deptDesc");    //获取部门简介
        test.insertDept(deptName,deptNum,deptDesc);
                            //调用 insertDept()方法添加部门
        return "success";       //返回请求操作成功
```

```
    }
    public String prepareUpdate(){       //得到部门 Id 为 deptId 的部门对象
        Test test = new Test();
        HttpServletRequest request=ServletActionContext.getRequest();
        int deptId=Integer.valueOf(request.getParameter("id"));
        Dept dept=test.prepareUpdateDept(deptId);
        request.setAttribute("dept", dept);
        return "success";
    }
    public String updateDept(){                //修改部门信息
        Test test = new Test();
        HttpServletRequest request=ServletActionContext.getRequest();
                                                //获取 request 对象
        int deptId=Integer.valueOf(request.getParameter("#request.dept.deptId"));
                                                //从 request 对象中取出部门 Id
        String deptName=request.getParameter("#request.dept.deptName");
                                                //从 request 对象中取出部门名称
        int deptNum = Integer.valueOf(request.getParameter
        ("#request.dept.deptNum"));
                                                //从 request 对象中取出部门编号
        String deptDesc = request.getParameter("#request.dept.deptDesc");
                                                //从 request 对象中取出部门描述
        test.updateDept(deptId, deptName, deptNum, deptDesc);
                                                //调用 updateDept()方法来修改部门信息
        return "success";                       //返回请求操作成功
    }
    public String deleteDept(){                 //删除一个部门信息
        Test test = new Test();                 //创建一个 Test 类对象
        HttpServletRequest request=ServletActionContext.getRequest();
                                                //获取 request 对象
        int deptId=Integer.valueOf(request.getParameter("id"));
                                                //从 request 对象中取出部门 Id
        test.deleteDept(deptId);
                                //调用 deleteDept()方法删除部门 Id 为 deptId 的部门信息
        return "success";                       //返回请求操作成功
    }

}
```

(4) 在项目的 src 目录下新建一个 struts.xml 文件，在该文件中添加 DeptAction 的配置，代码如下所示。

```
<struts>
    <include file="struts-default.xml"/>   <!-- 引入 struts-default.xml 文件 -->
    <package name="struts2" extends="struts-default">      <!-- 创建一个名为
struts2 的包 -->
        <!-- 配置一个名为 selectDept 的 Action,当请求这个 Action 时执行 DeptAction 中
的 selectDept 方法查询部门信息 -->
```

```
        <action name="selectDept" class="struts.org.db.DeptAction"
method="selectDept">
            <result name="success">dept.jsp</result><!-- 查询成功后,跳转到
dept.jsp -->
        </action>
        <!-- 配置一个名为 insertDept 的 Action,当请求这个 Action 时执行 DeptAction 中
的 insertDept 方法查询部门信息 -->
        <action name="insertDept" class="struts.org.db.DeptAction"
method="insertDept">
            <result name="success" type="redirect-action"><!-- 添加成功后,重定
向到 selectDept-->
                selectDept
            </result>
        </action>
        <!-- 配置一个名为 prepareUpdateDept 的 Action,当请求这个 Action 时执行
DeptAction 中的 prepareUpdate 方法查询部门信息 -->
        <action name="prepareUpdateDept"
        class="struts.org.db.DeptAction" method="prepareUpdate">
            <result>updateDept.jsp</result><!-- 获取要修改的部门信息成功后,跳转到
                                            updateDept.jsp -->
        </action>
        <!-- 配置一个名为 updateDept 的 Action,当请求这个 Action 时执行 DeptAction 中
        的 updateDept 方法查询部门信息 -->
        <action name="updateDept" class="struts.org.db.DeptAction"
method="updateDept">
            <result name="success" type="redirect-action"><!--修改部门信息成功
后,重定向到 selectDept -->
                selectDept
            </result>
        </action>
        <!-- 配置一个名为 deleteDept 的 Action,当请求这个 Action 时执行 DeptAction 中
        的 deleteDept 方法查询部门信息 -->
        <action name="deleteDept" class="struts.org.db.DeptAction"
method="deleteDept">
            <result name="success" type="redirect-action"><!-- 删除部门信息成功
后,重定向到 selectDept -->
                selectDept
            </result>
        </action>
    </package>
</struts>
```

(5) 前期工作都做好后，开始做显示操作部门信息的页面。在 Struts 2 App 项目的目录下新建一个 index.jsp 页面。代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
```

```
    <head>
        <title>查看部门信息</title>
    </head>
    <body>
        <s:form action="selectDept" method="post">     <!-- 创建一个表单,提交表
单发出 selectDept(查询部门信息的 Action 请求) -->
            <s:submit value="部门信息"/>
        </s:form>
    </body>
</html>
```

(6) 新建一个 dept.jsp 页面，用于显示部门信息，及执行“添加”、“修改”和“删除”部门信息的操作。代码如下所示。

```
<table align="center" width="70%" cellspacing="0" cellpadding="0" border=1>
    <tr align="center">
        <td>部门名称</td><td>部门人数</td><td>部门描述</td><td>操作</td>
    </tr>
    <s:iterator value="#request.depts" status="stuts"> <!-- 迭代显示当前所有部
门信息  -->
        <tr>
            <td><s:property value="deptName"/></td>
            <td><s:property value="deptNum"/></td>
            <td><s:property value="deptDesc"/></td>
            <td align="center">
                <!-- 添加修改当前行所代表的部门信息链接 -->
                <a href="prepareUpdateDept.action?id=<s:property
value='deptId'/>">修改</a>
                <!-- 添加删除当前行所代表的部门信息链接 -->
                <a href="deleteDept.action?id=<s:property value='deptId'/>">
删除</a>
            </td>
        </tr>
    </s:iterator>
    <tr>
        <!-- 添加新增部门信息链接 -->
        <td colspan=8><a href="<s:url value="/deptNew.jsp"/>">新增</a></td>
    </tr>
</table>
```

(7) 新建一个修改部门信息的 updateDept.jsp 页面，代码如下所示。

```
<s:form action="updateDept" method="post">
    <s:textfield name="#request.dept.deptId" label="部门 Id" readonly="true"/>
    <s:textfield name="#request.dept.deptName" label="部门名称"/>
    <s:textfield name="#request.dept.deptNum" label="部门编号"/>
    <s:textarea name="#request.dept.deptDesc" rows="5" cols="20" label="部门
简介"></s:textarea>
    <s:submit value="提交"></s:submit>
</s:form>
```

(8) 新建一个添加部门信息的 deptNew.jsp 页面，代码如下所示。

```
<s:form action="insertDept" method="post">
    <s:textfield name="deptName" label="部门名称"/>
    <s:textfield name="deptNum" label="部门编号"/>
    <s:textarea name="deptDesc" rows="5" cols="20" label="部门简介"/>
    <s:submit value="添加"></s:submit>
    <s:reset value="重置"></s:reset>
</s:form>
```

2.4.4 运行结果

打开 IE 浏览器，在地址栏中输入“http://localhost:8080/Struts2App/index.jsp”，访问查看部门信息页面，执行效果如图 2-6 所示。

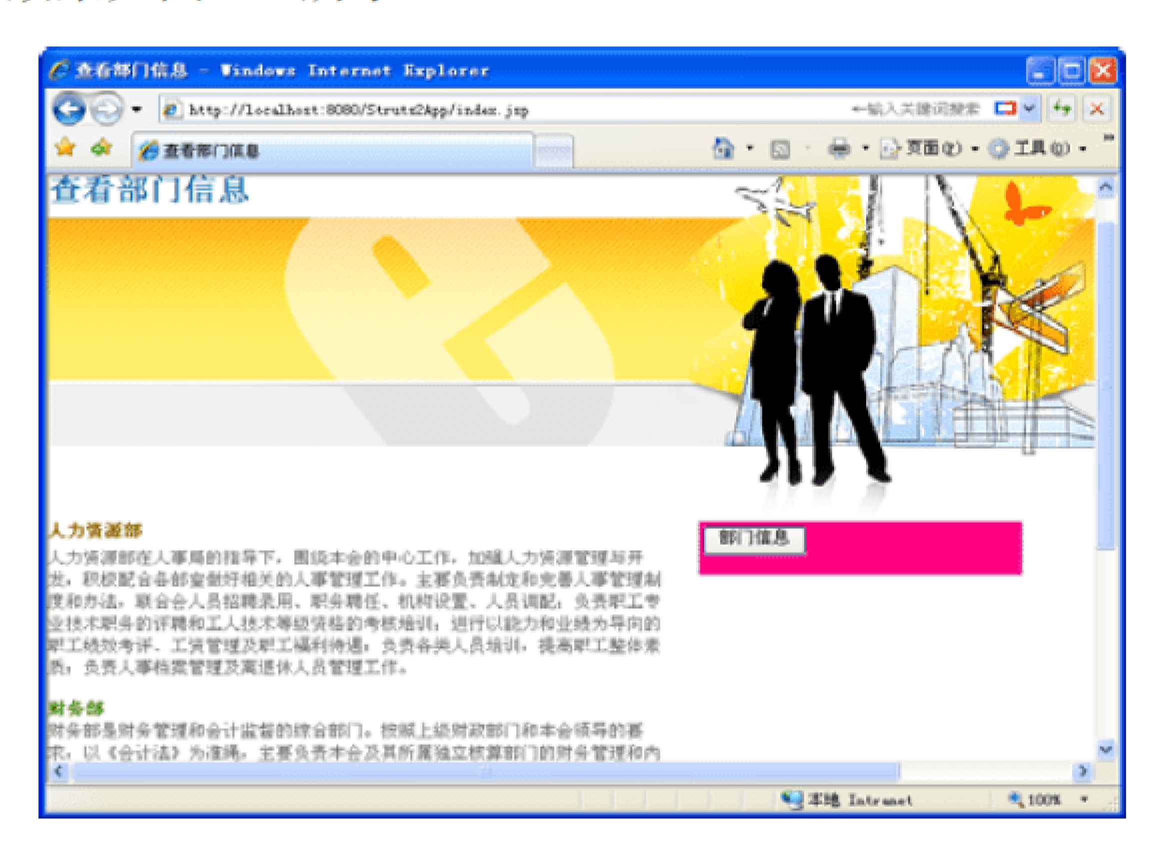


图 2-6 部门信息管理

单击查看部门信息页面中的“部门信息”按钮，即可查看当前所有部门的信息，还可以执行“添加”、“修改”和“删除”部门操作。执行效果如图 2-7 所示。



图 2-7 部门信息管理

单击“新增”链接，进入添加部门信息页面，从中可以输入新部门的信息，单击“添加”按钮，添加成功后，执行效果如图 2-8 所示。

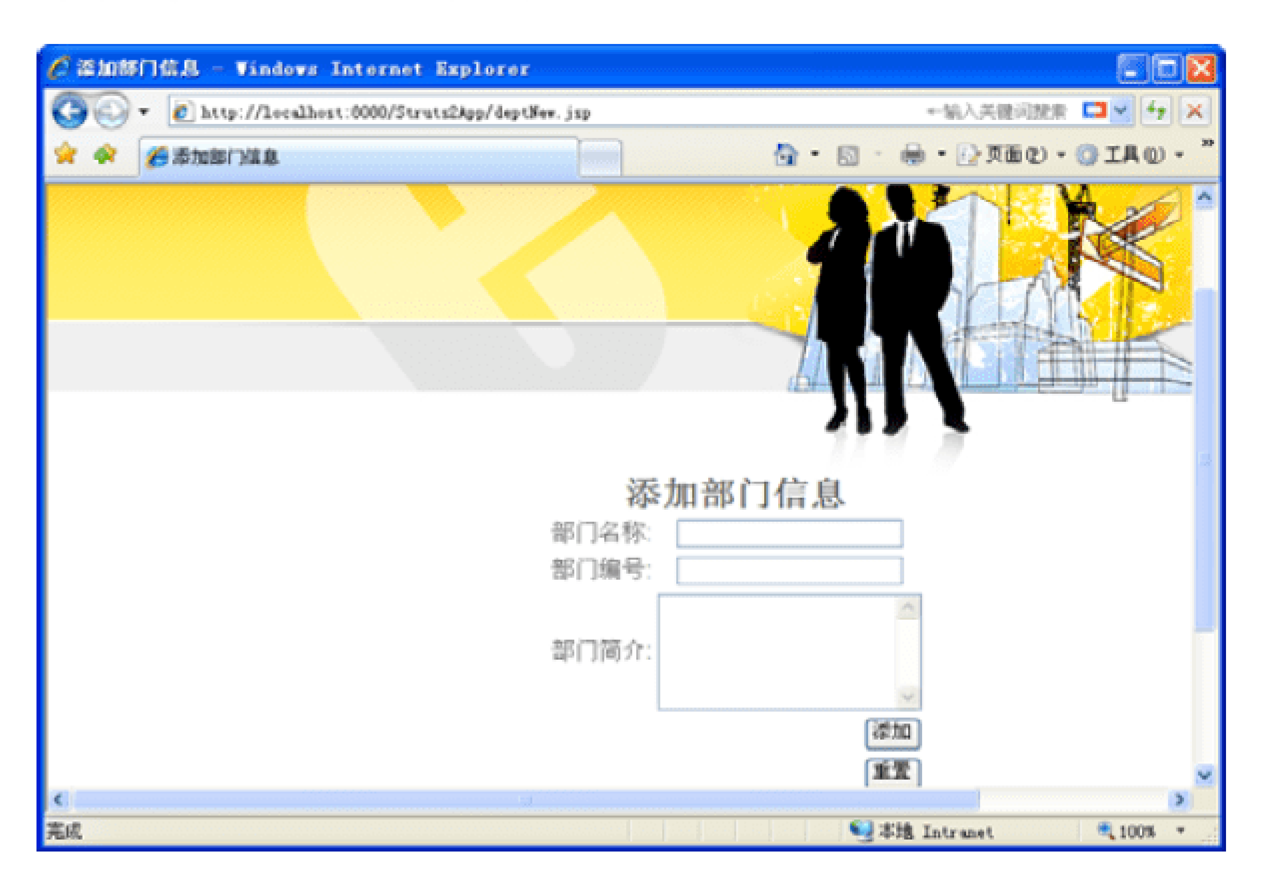


图 2-8　添加部门信息

“修改”和“删除”操作读者可以在做练习实例时执行查看效果，在此不再详解。

2.4.5　实例分析



源码分析：

本案例是对部门信息的管理，所以首先封装出一个部门实体类 Dept.java，包含部门 Id、名称、编号等。

然后定义一个部门 DeptAction，用于接收处理对部门信息的“增、删、查、改”操作的请求，而且在 struts.xml 文件中配置了 DeptAction。

最后创建一个前台操作页面，如：查看部门信息页面(index.jsp)、部门信息管理页面(dept.jsp)、添加部门信息页面(deptNew.jsp)、修改部门信息页面(updateDept.jsp)。

2.5　用户注册动态配置 Result

什么是动态配置 Result？动态就是有不确定因素，也就是说只有在执行的时候才能知道 Result。本节将详细介绍 Struts 2 是如何动态配置 Result 的。

视频教学： 光盘/videos/02/ dongtai_result_1.avi　　长度：7 分钟
光盘/videos/02/ dongtai_result_2.avi　　长度：9 分钟

2.5.1 基础知识——动态配置 Result

给大家举个例子，请读者想象一下状态机一样的执行流，下一个状态依赖于输入元素、session 属性、用户角色等的组合。换句话说，判定下一个 Action、输入和其他什么内容，这些在配置的时候是难以知晓的。

就像 Struts 2 中的标签库，通过使用 EL 表达式可以访问 Action 属性，也可以使用相应的方式访问 Result 的值。下面是 FragmentAction 的实现代码。

```
import com.opensymphony.xwork2.ActionSupport;
public class FragmentAction extends ActionSupport{
    private String nextAction;
    public String getNextAction() {
        return nextAction;
    }
}
```

可能会定义一个 Result，配置如下所示。

```
<action name="fragment" class="FragmentAction">
    <result name="next" type="redirectAction">${nextAction}</result>
</action>
```

如果 FragmentAction 返回的 Result 与 nextAction 属性的实际值相同。那么，nextAction 将把结果传递给下一个 redirectAction。

2.5.2 实例描述

Action 配置结果都是在配置文件中显现静态的，那么有没有可以动态配置 Result 的呢？当然有了，本节将通过用户注册来动态地配置 Result。

2.5.3 实例应用

【例 2-5】 用户注册动态配置 Result。

(1) 先在项目目录下创建一个用户注册页面 register.jsp，页面主要包含一个用户注册表单，需填入用户名、密码、性别信息，单击“现在注册”按钮即可注册。主要代码如下所示。

```
<FORM action="user!add" method="post" >
    <DIV class=logbox>
    <H1>您还没有注册!</H1>
    <UL>
        <LI>用户名  
            <INPUT type="text" name="userName">
        </LI>
        <LI>密    码  
```

```
            <INPUT type=password name="password">
        </LI>
        <LI>性    别  
            <INPUT type="text" name="sex">
        </LI>
        <DIV align=center><A style="COLOR: red"  href="">我要登录

    </A>           &
nbsp;
        <A class=numlink href="">找回密码! </A>
        </DIV>
    <DIV class=clear></DIV></LI></UL></DIV><!--登录-->
    <DIV class=lt id=jinru>
        <INPUT id=url type=hidden value=http://blogs.bokee.com/ name=url>
        <INPUT class=logbtn type=submit value=现在注册 name=submit>
    </DIV>
    <DIV class=clear>
    </DIV>
    </DIV>
</FORM>
```

(2) 创建一个用户注册 RegisterAction，用来处理用户注册操作，代码如下所示。

```
package cn.edu.ahau.mgc.struts2.action;
import com.opensymphony.xwork2.ActionSupport;
public class RegisterAction extends ActionSupport {

    private String userName;
    private String password;
    private String url;
    private String sex;
    public String getUserName() {
        return this.userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }
    //下面是和上面 getUserName()、setUserName()方法一样的用来获取和设置密码、
    //性别、url 的 get、set 方法
    public String add() {
        if (userName == null || "".equals(userName)||
        password ==null||"".equals(password)) {
            url = "/registerError.jsp";
        }
        else {
            url = "/registerSuccess.jsp";
        }
        return SUCCESS;
```

```
    }
}
```

(3) 定义好 RegisterAction 后，在 struts.xml 文件中配置这个 Action，代码如下所示。

```
<constant name="struts.devMode" value="true" />
<constant name="struts.i18n.encoding" value="UTF-8"></constant>
<package name="user" namespace="/" extends="struts-default">
    <action name="user"
class="cn.edu.ahau.mgc.struts2.action.RegisterAction">
        <result>${url}</result>
    </action>
</package>
```

(4) 创建一个用户注册成功提示页面 registerSuccess.jsp，代码如下所示。

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>Dynamic</title>
    </head>
    <body>
        恭喜您，注册成功!
    </body>
</html>
```

(5) 创建一个用户注册失败提示页面 registerError.jsp，代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>Dynamic</title>
    </head>
    <body>
        很抱歉，您注册失败!
    </body>
</html>
```

2.5.4 运行结果

打开一个浏览器页面，在地址栏中输入“http://localhost:8080/Struts2_2/register.jsp”运行注册页面，效果如图 2-9 所示。



图 2-9　用户注册动态配置 Result

当用户输入合法信息时，单击“现在注册”按钮，将会跳转到 registerSuccess.jsp 页面，不合法，将跳转到 registerError.jsp 页面。由于这两个提示页面比较简单，在此执行效果图不再显示。

2.5.5　实例分析



源码分析：

上述实例首先给出了一个用户注册页面，主要包含一个用户输入用户名、密码、性别等注册信息的表单，但用户输入信息完毕单击“现在注册”按钮后，将会把表单信息提交给 RegisterAction。这个 Action 中包含了一个特别重要的属性“url”，这个属性就是配置动态 Result 时要使用的，当判断用户输入的信息合法时，将这个属性赋值为“registerSuccess.jsp”，不合法时，将这个属性赋值为“registerError.jsp”，在 struts.xml 文件中采用<result>标签配置该 Action 的跳转结果时，应这样配置<result>${url}</result>。

2.6　登录异常处理

任何成熟的 MVC 框架都应该提供成功的异常处理机制。Strut 2 也不例外。Struts 2 提供了一种声明式的异常处理方式，它是通过配置拦截器来实现异常处理机制的。

视频教学：光盘/videos/02/exception.avi　　　　长度：7 分钟

2.6.1 基础知识——Struts 2 的异常机制

Struts 2 的异常处理机制通过在 struts.xml 文件中配置<exception-mapping…>元素完成，配置该元素时，需要指定两个属性。

- exception：此属性指定该异常映射所设置的异常类型。
- result：此属性指定 Action 出现该异常时，系统转入 Result 属性所指向的结果。

异常映射也分为两种。

- 局部异常映射：<exception-mapping…>元素作为<action…>元素的子元素配置。
- 全局异常映射：<exception-mapping…>元素作为<global-exception-mappings>元素的子元素配置。

输出异常使用 Struts 2 的<s:property>标签来实现，代码如下所示。

```
<s:property value="exception.message"/>    <!-- 输出异常对象本身 -->
<s:property value="exceptionStack"/>        <!--输出异常堆栈信息 -->
```

2.6.2 实例描述

相信学习编程的人对异常处理都不陌生，从刚开始学习 Java 编程语言时，就已经开始学习异常处理了，因为这是编程必用的知识。只有熟悉灵活地处理异常，才能使程序更加的完善，使用户使用起来也更加的得心应手。记得刚开始学习 Java 语言时，要求写个程序进行除法运算。由于当时还没学到异常处理，当输入除数为 0 时，虚拟机发出异常了，程序就死那儿了。

所以程序当中的异常处理是非常重要的，本节通过处理用户登录异常，来演示 Struts 2 的异常处理机制。

2.6.3 实例应用

【例 2-6】 登录异常处理。

(1) 在项目 Struts2_2 的 src 目录下，新建一个 lee 包，在该包下新建一个 LoginAction，用来处理用户的登录请求，代码如下所示。

```
package lee;
import com.opensymphony.xwork2.Action;
public class LoginAction implements Action
{
    private String username;
    private String password;
    private String tip;

    //下面是 username、password、tip 几个属性的 get、set 方法，在此省略
    …
```

```
    public String execute() throws Exception
    {
        if (getUsername().equalsIgnoreCase("user"))
        {
            throw new MyException("自定义异常");
        }
        if (getUsername().equalsIgnoreCase("sql"))
        {
            throw new java.sql.SQLException("用户名不能为 SQL");
        }
        if (getUsername().equals("scott") && getPassword().equals("tiger") )
        {
            setTip("哈哈,服务器提示! ");
            return SUCCESS;
        }
        else
        {
            return ERROR;
        }
    }
}
```

(2)　LoginAction 定义好后，在 struts.xml 文件中对其进行配置，代码如下所示。

```
<package name="mypackage" extends="struts-default">
    <!-- 此处定义所有的全局结果 -->
    <global-results>
        <result name="sql">/exception.jsp</result>
        <result name="root">/exception.jsp</result>
    </global-results>
    <!-- 此处定义全局异常映射 -->
    <global-exception-mappings>
    <!-- Action 抛出 SQLException 异常时,转入名为 sql 的结果 -->
    <exception-mapping result="sql"
        exception="java.sql.SQLException">
    </exception-mapping>
    <exception-mapping result="root"
        exception="java.lang.Exception">
    </exception-mapping>
    </global-exception-mappings>
    <action name="login" class="lee.LoginAction">
        <!-- 下面配置一个局部异常映射,当 Action 抛出 lee.MyException 时,转入名为 my 的
结果 -->
        <exception-mapping result="my"
            exception="lee.MyException">
        </exception-mapping>
        <!-- 下面定义结果 -->
        <result name="my">/exception.jsp</result>
        <result name="success">/welcome.jsp</result>
        <result name="error">/error.jsp</result>
```

```
    </action>
</package>
```

(3) 定义一个异常类 MyException.java，代码如下所示。

```
package lee;

public class MyException extends Exception
{
    public MyException(){}
    public MyException(String msg)
    {
        super(msg);
    }
}
```

(4) 在项目目录下新建一个 login_e.jsp 页面，页面中定义一个 form 表单，包含两个输入文本框和一个按钮，用户可以输入用户名与密码，单击“登录”按钮进行登录。代码如下所示。

```
<%@ page language="java" contentType="text/html; charset=GBK"%>
<html>
<head><title>登录页面</title></head>
<body>
    <h2 class="mem"><span>会员登录</span></h2>
    <form id="login" method="post" action="login.action">
        <div>
            请输入用户名:
            <input type="text" name="username" value="" class="txtBox" />
            请输入密码:
            <input type="password" name="password" value="" class="txtBox" />
        </div>
        <div id="foget">忘记密码?
        <input type="submit" name="login" value="登录" class="login"  />
        </div>
    </form>
</body>
</html>
```

(5) 新建一个欢迎页面 welcome.jsp，当用户登录成功时，跳转到该页面，代码如下所示。

```
<%@ page language="java" contentType="text/html; charset=GBK"%>
<%@taglib prefix="s" uri="/struts-tags"%>
<html>
    <head>
        <title>成功页面</title>
    </head>
    <body>
        您已经登录!
        <s:property value="model.tip"/>
    </body>
</html>
```

(6)　当用户登录出现异常时，需要处理异常，所以需要新建一个 exception.jsp 页面，代码如下所示。

```
<%@ page language="java" contentType="text/html; charset=GBK"%>
<%@taglib prefix="s" uri="/struts-tags"%>
<html>
    <head>
        <title>异常处理页面</title>
    </head>
    <body>
        <s:property value=" exception.message"/>
    </body>
</html>
```

(7)　当登录出现错误时，新建一个 error.jsp 页面即可解决，代码如下所示。

```
<%@ page language="java" contentType="text/html; charset=GBK"%>
<html>
    <head>
    <title>错误页面</title>
    </head>
    <body>
        您不能登录！
    </body>
</html>
```

2.6.4　运行结果

以下是登录异常捕获处理的过程，运行结果如图 2-10 所示。当用户单击“登录”按钮，会出现异常信息，如图 2-11 所示。



图 2-10　登录异常



图 2-11　异常处理

2.6.5 实例分析



源码分析：

当用户输入不合法的用户名时，会出现登录异常。通过<exception-mapping>标签配置一个局部异常映射，当 Action 抛出 lee.MyException 时，转入名为 my 的结果。使用<result name="my">/exception.jsp</result>标签指定 my 结果，跳转到 exception.jsp 页面，处理登录异常。

2.7 常见问题解答

2.7.1 Struts 2 配置常见异常处理



Struts 2 配置常见异常怎么处理？

网络课堂：http://bbs.itzcn.com/thread-10930-1-1.html

各位前辈，我初学 Struts 2，有很多都不是很明白，就比葫芦画瓢做了一个例子，我使用的开发环境如下。

```
JDK6+Tomcat6+Struts-2.1.6(最新)
Struts2 下载地址：http://struts.apache.org/
```

小程序用到 jar 包如下。

```
struts2-core.jar，commons-logging.jar，freemarker.jar，ognl.jar，xwork.jar
```

安装实例配置好以后，开始运行，可是出现以下错误。

```
严重：Exception starting filter struts2
Unable to load configuration. - bean -
jar:file:/E:/struts2/struts2/WebRoot/WEB-INF/lib/struts2-core-2.1.6.jar!
/struts-default.xml:46:178
```

【解决办法】 这个问题不是什么难解决的问题，只是你不知道，在 Struts 2 的 org.apache.struts2.dispatcher.multipart 包下，有个 JakartaMultiPartRequest 类，它需要引入下面如下几个类。

```
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.RequestContext;
import org.apache.commons.fileupload.disk.DiskFileItem;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
```

你的错误就是你没有加载这个几个类文件包，所以找不到类，只要把 commons-fileupload-1.2.1. jar 和 commons-io-1.3.2.jar 导入，问题就解决了。

2.7.2　HTTP Status 404 –在 Action 配置中没有找到相应的 Action Name 怎么办



HTTP Status 404–在 Action 配置中没有找到相应的 Action Name 怎么办?
网络课堂：http://bbs.itzcn.com/thread-10933-1-1.html

控制台总是输出如下错误。

```
HTTP Status 404 - There is no Action mapped for action name2010-03-05 15:37There
is no Action mapped for namespace / and action name ...
```

【解决办法】 对于这个问题确实不好解决，你可以从以下几种情况入手解决。

(1) 首先确认 struts.xml 文件是否在 src 目录下。很多时候 struts.xml 放在 WEB-INF 下而不是放在 classes 下。

(2) 检查 struts.xml 文件的语法是否正确。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 2.0//EN" "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <include file="struts-default.xml"/>
    <package name="struts2" extends="struts-default">
        <action name="login" class="com.test.action.LoginAction">
            <result name="success">/result.jsp</result>
        </action>
    </package>
</struts>
```

查看是否把 struts-default 中间的“-”错写成了 struts=default；或者把 struts 写成 sturts 等拼写。

(3) 还有其他一切正常，却把配置文件 struts.xml 错误地存储为 struts2.xml。

(4) 用 MyEclipse 开发直接部署到 Tomcat 下面，发现 MyEclipse 在部署的时候没有把 struts.xml 配置到 Tomcat 到 webapps 的对应项目下面，手工添加 struts.xml 后，即可解决问题！

2.7.3　Struts 2 Tomcat 6 MyEclipse 6.5 报 404 错误



Struts 2 Tomcat 6 MyEclipse 6.5 报 404 错误?
网络课堂：http://bbs.itzcn.com/thread-10934-1-1.html

使用 MyEclipse 6.5 部署 Struts 2 做的东西到 Tomcat6 上总是报 404 错误。为什么？不管哪个页面都访问不到。

jdk5、jvm6、struts2.1.6 jar 包都放好了的。web.xml 文件中也加入了 org.apache.struts2.dispatcher.FilterDispatcher。struts.xml 文件配置如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <include file="struts-default.xml"/>
    <package name="struts2 helloworld" extends="struts-default">
        <action name="Login" class="com.telshop.struts2.action.LoginAction">
            <result name="error">/error.jsp</result>
            <result name="success">/welcome.jsp</result>
        </action>
    </package>
</struts>
```

【解决办法】 首先看你的服务端报什么错？就是 Tomcat 上面的报错情况。可能很多人都遇到过这种问题，当配置好虚拟路径时，工程完全可以正常访问，但是当在 web.xml 中配置完 Struts 2 之后，突然就会报出 404 错误，页面已经无法找到了，而且 Tomcat 中没有任何的报错信息。

这到底是为什么呢？

答案很简单：版本问题！

如果你使用的是 Tomcat5.0，很抱歉，不支持！

我推荐你使用 Tomcat5.5 再重新试一次看看。

同时，Struts 2 不支持 JSP 标签。

如果你是一个刚接触 Struts 2 不到一天的新人的话，可以按照下面的方式试一下，也许会有不错的学习收获呢。

项目开发实践如下。

(1) 在 Eclipse 中建立一个 Web Project，并且向 WEB-INF/lib 中加入所需 Struts 2 的 5 个 jar 包(可从 Apache 的官方网站上下载)。

```
struts2-core.jar
xwork.jar
ognl.jar
freemarker.jar
commons-logging.jar
```

(2) 配置 web.xml，代码如下所示。

```
<!--FilterDispatcher 类在 struts2-core.jar 包中-->
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-cl
ass>
</filter>
<filter-mapping>
```

```
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<!--添加欢迎页面-->
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

(3)　创建类 HelloStruts，需继承 ActionSupport.java。

```
package org.bixy.struts2.demo;
import com.opensymphony.xwork2.ActionSupport;
public class HelloStruts extends ActionSupport {
    private String meg="hello,world!";
    public String getMeg() {
        return meg;
    }
    public void setMeg(String meg) {
        this.meg = meg;
    }
@Override
    public String execute() throws Exception {
        String f="";
        if("hello".equals(meg))
        {
            f="suc";
        }else{
            f="err";
        }
        return f;
    }
}
```

(4)　在 src 文件夹下创建 struts.xml 文件，配置如下。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <include file="struts-default.xml"/>
    <package name="bixy" namespace="/" extends="struts-default">
        <action name="hello" class="org.bixy.struts2.demo.HelloStruts">
            <result name="suc">
                /hello.jsp
            </result>
            <result name="err">
                /error.jsp
            </result>
        </action>
```

```
    </package>
</struts>
```

(5) 在 index.jsp 中编写提交表单的信息，代码如下。

```
<%@ page language="java" pageEncoding="gbk"%>
<body>
    <form action="hello.action" method="post">
        name:<input name="meg" type="text">
        <input type="submit" value="submit"/>
    </form>
</body>
```

(6) 在 hello.jsp 中编写显示输入信息的代码，如下所示。

```
<%@ page language="java" pageEncoding="gbk"%>
<%@ taglib prefix="s" uri="/struts-tags" %><!--引入的 struts2 标签-->
<body>
    输入的内容为：<s:property value="meg"/>
</body>
```

(7) 在 error.jsp 中编写出错时的信息，代码如下。

```
<%@ page language="java" pageEncoding="gbk"%>
<body>
输入内容有误！
</body>
```

如果你已经做到这一步了，不妨运行一下看看，执行效果应该不错。

2.7.4 Struts 2 配置问题 Error filterStart 如何解决



Struts 2 配置问题 Error filterstart 如何解决？
网络课堂：http://bbs.itzcn.com/thread-10936-1-1.html

请各位帮忙，我的 Struts 2 配置报如下错误。

```
org.apache.catalina.core.StandardContext start
严重: Error filterStart
2010-5-5 9:07:13 org.apache.catalina.core.StandardContext start
严重: Context [/TheLIFE] startup failed due to previous errors
2010-5-5 9:07:13 org.apache.coyote.http11.Http11Protocol start
信息: Starting Coyote HTTP/1.1 on http-8080
2010-5-5 9:07:14 org.apache.jk.common.ChannelSocket init
信息: JK: ajp13 listening on /0.0.0.0:8009
2010-5-5 9:07:14 org.apache.jk.server.JkMain start
信息: Jk running ID=0 time=0/16  config=null
2010-5-5 9:07:14 org.apache.catalina.startup.Catalina start
信息: Server startup in 5567 ms
```

报错信息使我无从入手，因为我的驱动包正确加载，工程也正确加载。xml 文件配置是按着书上写也没问题。在 Tomcat 中的 manage 工程始终无法启动。

【解决办法】：出现这种错误，一般有以下几种原因。

(1) xml 配置失误。filter 应配置在 servlet-mapping 前面(应该都知道吧)。观察 class 配置是否能找得到。

(2) filter 中某段代码未实例化(这个情况出现得最多，要仔细检查)。

(3) 试着把 tomat/server/lib 目录下的 commons-digester.jar,commons-beanutils.jar 复制到 common/lib/目录下，因为有些包在部署时没有被包含。

(4) 类文件可能没有部署到 tomcat 下面，致使清除掉了整个工程，建议重新编译部署，启动 Tomcat 再试一下。

2.8 习　　题

一、填空题

(1) 使用<filter-class>标签配置 Struts 2 核心 Filter 的实现类____________。

(2) include 节点是 Struts2 中组件化的方式，可以将每个功能模块独立到一个 xml 配置文件中，然后用 include 节点引用。通过____________在 struts.xml 文件中引入 struts-default.xml。

(3) <package>标签的____________属性，可用来指定包的命名空间。

(4) 自定义的 Action 必须继承 ActionSupport 类，而且需要重写____________方法。

(5) Result 常见的类型有 redirect、chain、____________。

(6) Struts2 的异常处理机制是通过在 struts.xml 文件中配置____________元素完成的。

二、选择题

(1) <result>标签中的 type 属性配置为____________重定向一个 URL。

A. chain　　B. redirect　　C. dispatcher　　D. redirectAction

(2) 在 struts.xml 文件中配置常量是一种指定 Struts2 属性的方式。我们使用____________来配置常量。

A. <init-param>　　B. <action>　　C. <bean>　　D. <constant>

(3) 输出异常使用 Struts2 的<s:property>标签来实现，但具体输出异常堆栈，应使用代码____________。

A. <s:property value="exceptionStack"/>

B. <s:property value="exception"/>

C. <s:property value="message"/>

D. <s:property value="exception.message"/>

(4) Bean 元素的 scope 属性是配置 Bean 实例的作用域。下面哪些是该属性可配置的值？____________

A. response　　B. single　　C. session　　D. page

三、上机练习

上机练习：实现一个添加并显示商品信息的实例。

本案例定义了一个商品类 Product.java，并创建了一个 ProductConfirm 类，它是一个 Action，这个 Action 获取用户通过 AddProduct.jsp 页面添加的商品信息，添加成功之后跳转到 ShowProduct.jsp 页面，该页面通过<s:iterator>标签将添加的商品依次显示在页面上。

最终运行后添加商品效果如图 2-12 所示，显示商品信息如图 2-13 所示。

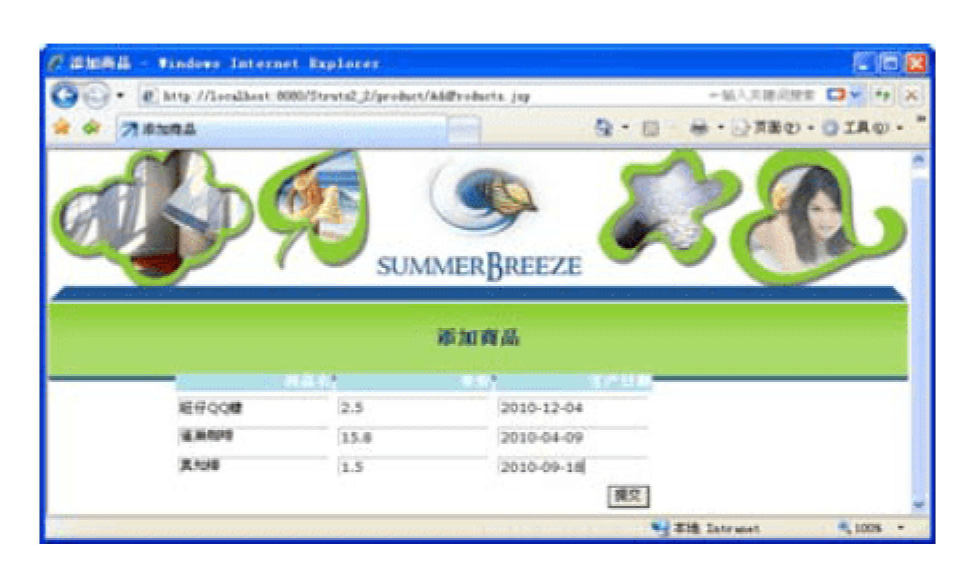


图 2-12　添加商品



图 2-13　显示商品信息



第 3 章　数据类型大转换

内容摘要：

HTTP 协议中，传递的任何内容都是 String 类型。一旦需要非 String 类型的对象，例如 int 或者 Date，就需要在收到 HTTP 请求的数据时，首先将 String 类型的数据变换为对应类型的数据，然后再使用该数据。这个变换过程就是类型转换。

Struts 2 提供了非常强大的类型转换机制，Struts 2 的类型转换是基于 OGNL 表达式的，只要把 HTML 输入项(表单元素和其他 GET/POST 的参数)命名为合法的 OGNL 表达式，就可以充分利用 Struts 2 的类型转换机制。

本章将介绍 Struts 2 中的类型转换及转换异常处理。

学习目标：

- 熟悉类型转换的作用。
- 熟练使用类型转换器。
- 掌握 Struts 2 对 null 属性的处理。
- 掌握 Struts 2 的类型转换对 List、Map 和 Set 的支持。
- 熟练开发自定义的类型转换器。
- 掌握对类型转换错误的处理。
- 熟练使用注解来配置类型转换。

3.1 类型转换的意义

所有的 MVC 框架，都是表现层的解决方案，都需要负责收集用户请求参数，并将请求参数传给应用的控制器组件。此时问题出现了，所有的请求参数都是字符串数据类型，但 Java 是强类型的语言，因此 MVC 框架必须将这些字符串请求参数转换成相应的数据类型。本节将为大家介绍一下 MVC 中表现层(View)的数据类型转换。

视频教学：光盘/videos/03/type.avi　　　　**长度：**15 分钟

3.1.1 基础知识——类型转换的意义

对于一个成熟的、智能的 MVC 框架而言，不可避免地需要实现类型转换。因为 B/S(浏览器/服务器)应用与传统的 C/S(客户端/服务器)应用程序不同，B/S 结构应用的请求参数是通过浏览器发送到服务器的，这些参数不可能有丰富的数据类型，因此必须在服务器端完成数据类型转换。

MVC 框架是一个表现层的解决方案，理应提供类型转换的支持，Struts 2 提供了功能非常强大的类型转换支持。

对于 Web 应用而言，表现层主要用于与用户交互，包括收集用户输入数据，向用户呈现服务器状态数据。因此表现层数据的流向主要有两个方向：输入数据和输出数据。对于将服务器状态呈现给客户整个方向而言，不管是 Java 的输出语句，还是 JSP 的输出语句，都支持多种数据类型的直接输出，应用程序只需要使用简单的输出语句即可将服务器状态呈现给浏览器。对于输入数据，则需要完成由字符串数据向多种数据类型转换。程序通常无法自动完成数据类型转换，需要在代码中手动转换。表现层数据的流向和类型转换如图 3-1 所示。

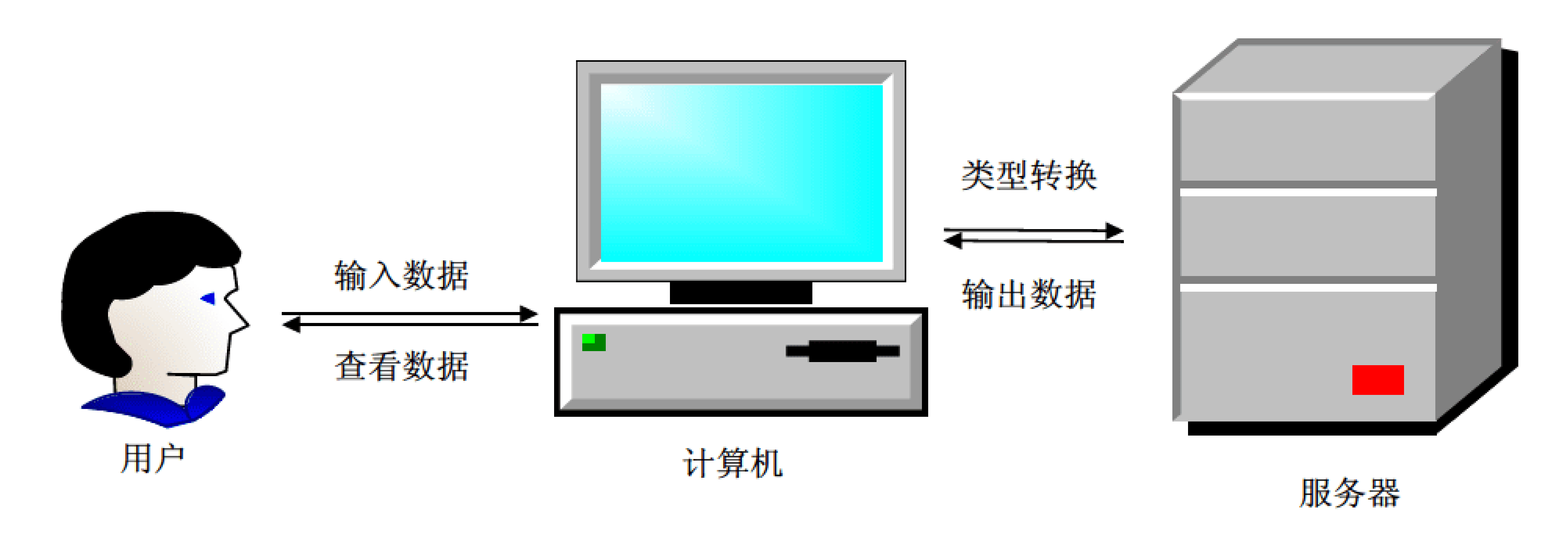


图 3-1　表现层数据的流向和类型转换

表现层的另一个数据处理是数据校验，数据校验可分为客户端校验和服务器端校验两种。客户端校验和服务器端校验都是必不可少的，二者分别完成不同的过滤。

关于数据校验的知识，将会在后面章节中详细介绍。

3.1.2　实例描述

还记得刚开始接触 Java Web 编程时获取表单元素吗？当初采用了 HttpServlet Request 的 getParameter(“元素 name”)方法来获取表单元素，而这种获取方法全部都是 String 类型，当输入一个年龄或者日期的时候，获取到的是一个只包含数字或者日期格式的字符串，但是 Bean 类中的年龄和出生日期分别是 int 类型和 java.util.Date 类型，怎么把获取的表单元素值赋给 Bean 的属性呢？这就需要用到类型转换。

以下的实例展示的是当时的做法。

3.1.3　实例应用

【例 3-1】 用户注册输入数据类型大转换。

(1) 新建 Struts 2-Converter 项目，在该项目下新建 com.struts2.bean 包，并在其下创建 UserBean 类，用于收集用户输入的注册信息。注册信息包括用户名、密码、年龄、出生日期。完整代码如下。

```
package com.struts2.bean;
import java.util.Date;
public class UserBean {
    private String name;//用户名
    private String pass;//密码
    private int age;//年龄
    private Date birth;//出生日期
    //无参构造函数
    public UserBean(){
    }
    //有参数的构造器
    public UserBean(String name,String pass,int age,Date birth){
        this.name=name;
        this.pass=pass;
        this.age=age;
        this.birth=birth;
    }
    /*下面是上面 4 个属性的 set、get 方法，这里省略*/
}
```

(2) 在 com.struts2.servlet 包下新建 Register.java，用于将页面收集到的数据封装成一个 UserBean 的值对象。该 Servlet 的代码如下。

```
package com.struts2.servlet;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.servlet.ServletException;
```

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.struts2.bean.UserBean;
public class Register extends HttpServlet {
    /**
     * Servlet 的服务响应方法
     */
    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        //设置解码格式
        request.setCharacterEncoding("GBK");
        //下面依次获取 4 个参数，获取的参数全部为字符串类型
        String name=request.getParameter("name");
        String pwd=request.getParameter("pass");
        String strAge=request.getParameter("age");
        String strBirth=request.getParameter("birth");
        //下面进行类型转换
        //字符串类型向整型转换
        int age=Integer.parseInt(strAge);
        //使用 SimpleDateFormat 将字符串向日期格式转换
        //指定转换格式，日期字符串，必须按 yyyy-MM-dd 的格式输入
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");
        Date birthDate=null;
        try {
            birthDate=sdf.parse(strBirth);
        } catch (Exception e) {
            e.printStackTrace();
        }
        //将类型转换后的值封装成 UserBean 的值对象
        UserBean userBean=new UserBean(name,pwd,age,birthDate);
        //用 Servlet 直接输出
        response.setContentType("text/html;charset=gb2312");
        //获得页面输出流
        PrintWriter out=response.getWriter();
        out.print("<html><head><title>");
        //输出页面标题
        out.println("类型转换页面");
        out.println("</title></head><body>");
        out.println("<h1>类型转换</h1>");
        out.println(userBean.getName()+"<br>");
        out.println(userBean.getPass()+"<br>");
        out.println(userBean.getAge()+"<br>");
        out.println(userBean.getBirth()+"<br>");
        out.println("</body></html>");
    }
    @Override
    public void destroy() {
```

```
        super.destroy();
    }
}
```

(3) 为了让上面的 Servlet 可以直接生成对浏览者的响应，必须将该 Servlet 配置在 Web 应用中，配置 Servlet 必须在 web.xml 文件中增加下面的代码片段。

```
    <servlet>
        <servlet-name>regist</servlet-name>
        <servlet-class>com.struts2.servlet.Register</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>regist</servlet-name>
        <url-pattern>/regist</url-pattern>
    </servlet-mapping>
```

(4) 创建用户注册页面 register.jsp。因为上面配置的 Servlet 映射的 URL 为/regist，故将表单页面的<form.../>元素的 action 属性设置为 regist，而且该表单将提交到 Servlet 类—Register 类中。代码片段如下。

```
    <form action="regist" method="post">
        username:<input type="text" name="name"><br>
        userpass:<input type="password" name="pass"><br>
        userage:<input type="text" name="age"><br>
        userbirth:<input type="text" name="birth"><br>
        <input type="submit" value="Regist" align="center">
    </form>
```

3.1.4　运行结果

运行 register.jsp 页面，出现如图 3-2 所示的界面。



图 3-2　注册信息的收集页面

在图 3-2 所示的页面中，依次在 username 文本框中输入 lizanhong；在 userpass 口令框中输入 123456；在 userage 文本框中输入 22；在 userbirth 文本框中输入 1989-02-21。单击 Regist 按钮，出现如图 3-3 所示的界面。

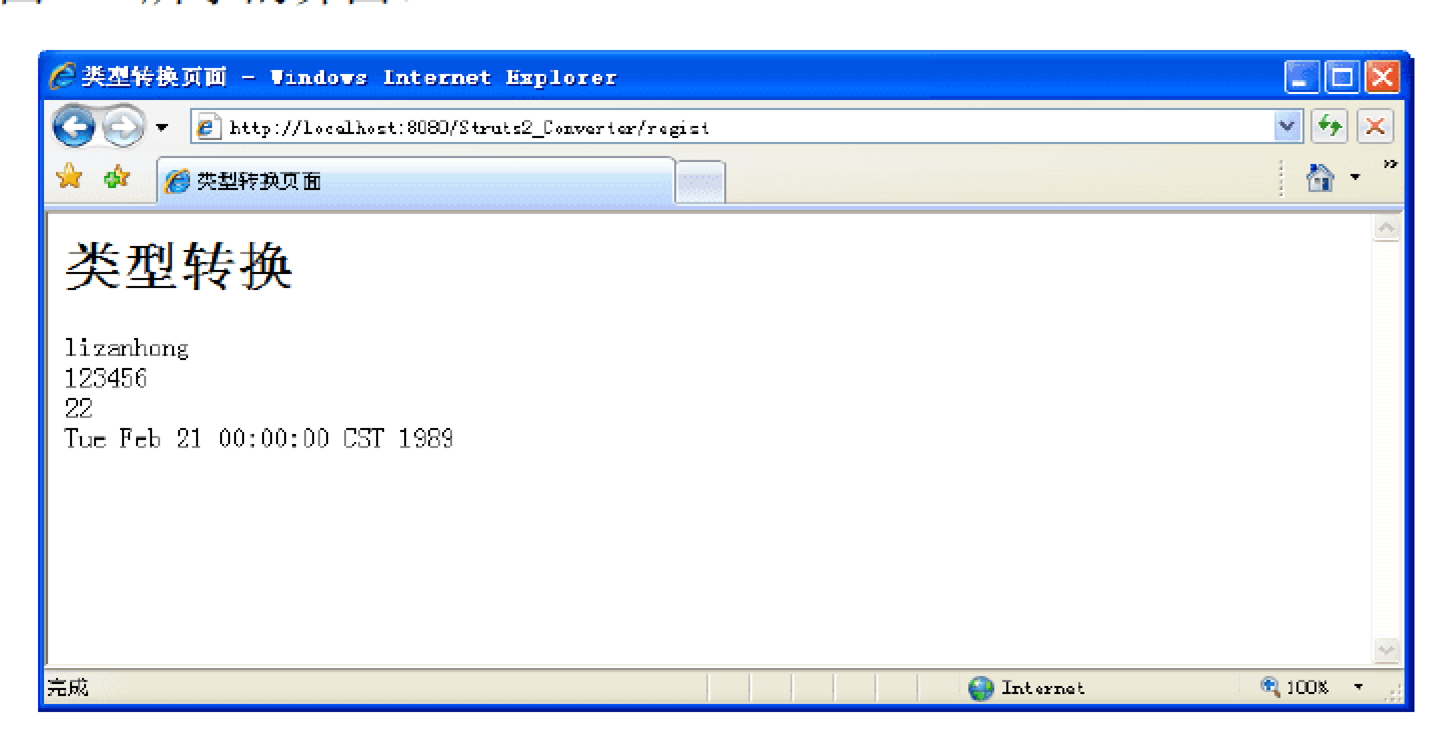


图 3-3　类型转换效果

3.1.5　实例分析



源码解析：

在上面的例子中，单击 Regist 按钮后，表单提交至 Register 类，即 Servlet 类中，在该 Servlet 类中获取用户输入的四个字符串参数，并经过类型转换后封装到一个 UserBean 的对象中。对于这个应用，程序员在 Servlet 中进行类型转换，并将其封装成请求参数的值对象由于这些类型转换操作全部手工完成，因此显得异常繁琐。

对于 MVC 框架而言，它一样需要将请求参数封装成 VO 对象。因此，必须把请求参数转换成 VO 类里各属性对应的数据类型——这就是类型转换的意义。

3.2　使用 Struts 2 的类型转换

在 B/S 应用中，将字符串请求参数转换为相应的数据类型，是 MVC 框架提供的功能。Struts 2 作为很好的 MVC 框架实现者，也提供了类型转换机制。



视频教学：光盘/videos/03/internal.avi　　长度：11 分钟

3.2.1　基础知识——Struts 2 对类型转换的支持

Struts 2 类型转换机制的基础是 OGNL 表达式，在前面章节的例子程序中，相信读者已经感受到了 Struts 2 强大的类型转换能力。

1．使用 OGNL 表达式命名参数

在用户注册 Action 中存在一个 user 属性(类型为 User)，并提供了 user 属性的 getter 和 setter 方法。在为表单输入元素命名时，只需要把它们命名为合法的 OGNL 表达式，例如：user.username、user.age、user.birthday 等，这样就可以使用 Struts 2 的类型转换功能了。Struts 2 框架会自动对输入数据按照 user 对象的属性的类型进行类型转换(如果 user 对象不存在，框架还会为此创建 user 对象)，并用转换后的数据封装 user 对象。

2．Struts 2 内置的类型转换器

Struts 2 框架可以自动转换常见的数据类型。这些转换工作，完全可以交给 Struts 2 系统实现，开发者不用再自定义类型转换器。常用的类型转换包括下列类型和 String 类型的相互转换。

1) String

将 int、long、double、boolean、String 类型的数组或 java.util.Date 类型转换为字符串。

2) boolean/Boolean

在字符串和布尔值之间进行转换。

3) char/Character

在字符串和字符之间进行转换。

4) int/Integer、float/Float、long/Long、double/Double

在字符串和数值型的数据之间进行转换。

5) date

在字符串和日期类型之间进行转换。对于日期类型，采用 SHORT 格式来处理输入和输出，使用当前请求关联的 Locale 来确定日期格式。

6) array

由于数组元素本身就有类型，Struts 2 使用元素类型对应的类型转换器，将字符串转换为数组元素的类型，然后再设置到新的数组中。

7) collection

如果不能确定对象类型，则假定集合元素类型为 String，并创建一个新的 ArrayList，存放所有的字符串。



对于数组的类型转换将按照数组元素的类型，尝试独立地转换每一项。和任何其他的类型转换一样，在类型转换的处理过程中，如果转换不能执行，标准的类型转换错误报告将被用于指出发生的问题。

3．null 属性的处理

当发现 null 引用时，null 属性处理将自动创建对象。如果 Action 上下文中存在键#CREATE_NULL_OBJECTS，并且其值为 true(这个键只在 com.opensymphony.xwork2.interceptor.ParametersInterceptor 执行期间被设置)，那么引发 NullPointerException 的 OGNL 表达式将被暂停求值。此时系统将通过创建所需对象的方法来自动尝试解决 null 引用。

处理 null 引用时将遵循下列规则。

- 如果属性声明为 Collection 或者 List，那么将创建一个 ArrayList 对象，并赋值给 null 引用。
- 如果属性声明为 Map，那么将创建一个 HashMap 对象，并赋值给 null 引用。
- 如果 null 属性是一个具有无参构造方法的简单 Bean，那么将使用 ObjectFactory 的 buildBean()方法创建一个 Bean 的实例。



例如，如果表单元素中有一个文本字段命名为 person.name，而表达式 person 计算结果为 null，那么 ObjectFactory 类将被调用。由于表达式 person 的类型为 Person 类，因此一个新的 Person 对象将被创建，并赋值给这个 null 引用。最后，name 值被赋给该对象的 name 属性。整个过程就是系统自动创建一个 Person 对象，通过调用 setPerson()方法将它赋给 null 引用，最后调用 getPerson().getName()来设置 name 属性，而这通常也是读者想要的结果。

4．局部类型转换器

上面介绍了几种常见的类型转换器，关于类型转换器的注册方式，主要有如下三种。

- 注册局部类型转换器：局部类型转换器仅仅对某个 Action 的属性起作用。
- 注册全局类型转换器：全局类型转换器对所有 Action 的特定类型的属性都会生效。
- 使用 JDK1.5 的注释来注册类型转换器：通过注释方式来生成类型转换器。

下面介绍的是局部类型转换器的注册方式，要注册类型转换器只需提供文件名为如下格式的文件。

```
ClassName-conversion.properties
```

其中，“ClassName”是需要转换器生效的类名，后面的“-conversion.properties”字符串是固定的。该文件是一个典型的 Properties 文件，文件由 key-value 对组成。文件内容如下所示。

```
propertyName=类型转换器类
```

ClassName-conversion.properties 文件由多个“propertyName=类型转换器类”项组成，其中“propertyName”是类中需要类型转换器转换的属性名，类型转换器类是开发者实现的类型转换器的全限定类名(需要加包前缀)。

例如，在用户 Action 类(UserAction)中有一个 User 动作类属性 user，同时又有 user 属性的转换器类 UserConverter 类来完成类型转换，可以命名类型转换器注册文件为 UserAction-conversion.properties，并在其内添加如下内容。

```
user=com.struts2.convert.UserConverter
```

UserAction-conversion.properties 文件应该与 UserAction.class 放在相同位置(因为 UserAction 的包为 com.struts2.actions，因此该文件应该放在 Web 应用的 WEB-INF/classes/com/struts2/actions 路径下)。

到此，局部类型转换器注册成功。当浏览者提交请求时，请求中的 user 请求参数将会先被该类型转换器处理。

局部类型转换器只对指定类的特定属性起作用，这具有很大的局限性——花费了大量的时间完成了一个类型转换器，却只能使用一次(对一个类有效)，这太浪费了。通常我们会将类型转换器注册成全局类型转换器，让该类型转换器对该类型的所有属性起作用。



局部类型转换器只对指定类的指定属性生效，全局类型转换器对指定类型的全部属性起作用。

5. 全局类型转换器

假设应用中有多个 Action 都包含了 User 类型的属性，如果多次重复注册局部类型转换器，将是很烦琐的事情。因此，Struts 2 提供了全局类型转换器，它对指定类型的全部属性有效。

下面是一个请求对应的 Action 的代码片段。

```
public class UserAction extends ActionSupport {
    //转换信息
    private String tip;
    //封装 user 请求参数的属性
    private User user;
    //封装 customer 请求参数的属性
    private User customer;
    //封装 birth 请求参数的属性
    private Date birth;
    //处理用户请求的 execute()方法
    @Override
    public String execute() throws Exception {
        if(getUser().getName().equals("admin")&&getUser().getPass().
equals("admin")){
            setTip("转换成功");
            return SUCCESS;
        }
        else {
            setTip("转换失败");
            return ERROR;
        }
    }
    /*下面是上面所有属性的 set、get 方法，这里省略*/
    …
}
```

在上面的 Action 中，user 和 customer 两个属性的类型都是 User 类型——对于一个 Action 里包含两个 User 类型属性的情况，可以使用一个局部类型转换器注册文件完成注册；但如果系统中的多个 Action 都包含 User 类型的属性，则应该使用全局类型转换器注册。

注册全局类型转换器提供一个 xwork-conversion.properties 文件，该文件也是 Properties 文件，其内容由多个“符合类型=对应类型转换器”项组成，其中“复合类型”指定需要完成类型转换的复合类，“对应类型转换器”指定所指定类型转换的转换器。

注册全局类型转换器的注册文件代码如下所示。

```
com.struts2.model.User=com.struts2.convert.UserConverter
```

一旦注册了上面的全局类型转换器，该全局类型转换器就会对所有 User 类型属性起作用。

6. Struts 2 对 Collection 和 Map 的支持

Struts 2 对集合类型的转换提供了很好的支持，可以用集合对象来保存表单提交的数据，这对于同时提交多个相同类别的信息将会非常方便。

1) 指定集合元素的类型

修改上面 Action 类代码，修改后的 Action 类代码片段如下。

```
public class UserAction extends ActionSupport {
    //声明 user 集合
    private List user;
    @Override
    public String execute() throws Exception {

    if((User)getUser().get(0)).getName().equals("admin")&&((User)getUser().
get(0)).getPass().equals("admin"))
        {
            …
        }
    }
    public List getUser() {
        return user;
    }
    public void setUser(List user) {
        this.user = user;
    }
}
```

显然，对于上面的 user 属性，系统根本不清楚 List 里元素的类型，类型转换器也就无法起到作用。因此，为了让系统指导 List 里的元素类型，为了让系统的类型转换器起到作用，我们可以用以下两种方法。

- 使用泛型来限制集合里元素的类型。
- 使用 Struts 2 的配置文件：使用局部类型转换的配置文件来制定集合元素的数据类型。

为了在局部类型转换文件中指定集合元素的类型，应该在局部类型转换文件中增加如下的 key-value 对。

```
Element_xxx=复合类型
```

上面 key-value 对中的“Element”是固定的，“xxx”是 Action 中的集合属性名，复合类型是集合元素类型的全限定类名(应该增加完整的包前缀)，例如修改前面所讲的局部类型转换器中的 UserAction-conversion.properties 文件内容(该文件内容只有一行代码)代码如下。

```
#指定 Action 类的 user 集合属性的元素为 com.struts2.model.User 实例
Element_user=com.struts2.model.User
```

增加定义后，系统将可以识别出 user 集合属性的集合元素是 com.struts2.model.User 类型，相应的类型转换器将可以对该属性生效。

对于 Map 类型的属性，则需要同时指定 Map 的 key 类型和 value 类型，为了指定 Map 类型属性的 key 类型，应该在局部类型转换文件中增加如下项。

```
Key_xxx=复合类型
```

其中"Key"是固定的，"xxx"是 Map 类型属性的属性名，复合类型指定的是 Map 属性的 key 类型的全限定类名。为了指定 Map 类型属性的 value 类型，应该在局部类型转换文件中增加如下项。

```
Element_xxx=复合类型
```

其中"Element"是固定的，"xxx"是 Map 类型属性的属性名，复合类型指定的是 Map 属性的 value 类型的全限定类名。

通过上面的局部类型转换元素指定集合元素类型后，系统的类型转换器将可以正常工作。为了在 JSP 页面中输出 List 属性的某个元素的值，可以使用如下格式。

```
<s:property value="集合属性名[索引].集合元素属性名"/>
```

下面是 JSP 页面中输出 Action 实例中属性的代码片段。

```
用户 1 的用户名为：<s:property value="user[0].name"/>
```



对于 Set 类型的属性是无法通过索引访问集合元素的，因为 Set 是无序集合。但是，Struts 2 提供了一个指定索引属性的方法可以访问 Set 类型的属性。

2)　指定集合元素的索引属性

为了更好地支持 Set 类型的属性，Struts 2 允许指定 Set 集合元素的 key 属性——该 key 属性可以唯一地标识该元素。假设有如下的 Action 类。

```
public class UserAction extends ActionSupport {
    //Action 包含了一个 Set 类型的属性，用于封装包含多个值的请求参数
    private Set user;
    /**
     * 处理用户请求的 execute()方法，直接返回 success 字符串
     */
    @Override
    public String execute() throws Exception {
        return SUCCESS;
    }
    public Set getUser() {
        return user;
    }
    public void setUser(Set user) {
        this.user = user;
    }
}
```

在上面的 Action 类中，包含了一个 Set 类型的属性，该属性没有使用泛型来限制集合元素的类型，因此应该在局部类型转换文件中指定集合元素的类型。除此之外，为了更方便地访问 Set 属性中的集合元素，需要指定集合元素的索引属性，指定集合元素的索引属性通过在局部类型转换文件中增加如下项即可。

```
Keyproperty_集合属性名=集合元素的索引属性名
```

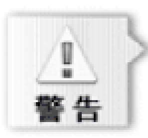
其中，上面的“集合属性名”是 Action 中的集合属性名(user)，集合元素的索引属性名是可以唯一标识集合元素的属性名。例如：

```
#指定 user 集合属性的集合元素类型是 com.struts2.model.User
Element user=com.struts2.model.User
#指定 user 集合属性里集合元素的索引属性是 name
KeyProperty_user=name
```

一旦指定了集合元素的索引属性值后，就可以直接通过该索引属性值来访问该集合元素，避免了只能直接访问有序集合里元素的缺陷。下面是在 JSP 页面中通过索引属性直接访问 Set 元素的代码片段。

```
用户 1 的用户名为：<s:property value="user('admin').name"/>
```

通过代码可以看出，直接访问 Set 元素的方式是：集合属性名('索引属性值')——该方式访问的是索引属性为指定值的集合元素。



上面访问 Set 元素用的是圆括号，而不是方括号。但对于数组、List 和 Map 属性来说，则需要通过方括号来访问指定集合元素。

3.2.2 实例描述

大家都做过各种各样的购物管理系统吧！也许只做过订购一件商品的管理系统；也许运用过 List 集合获取购买的物品信息 Z……学了 Struts 2 的类型转换，我们将采用 Map 类型来获取购买商品信息，认识一下 Struts 2 的类型转换在其中起着什么样的作用。没有了 Struts 2 的类型转换，系统是否还能正常运行呢？

下面将使用 Map 类型实现图书购物系统。

3.2.3 实例应用

【例 3-2】 使用 Map 实现图书购物系统。

(1) 在 Web 应用的 web.xml 文件中配置 Struts 2 的控制器 FilterDispatcher，配置代码如下。

```
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-cl
ass>
</filter>
<filter-mapping>
```

```
    <filter-name>struts2</filter-name>
    <url-pattern>*.action</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

(2)　编写 Book 类。Book 类是一个 JavaBean 类，是图书信息的对象化标识。在 src 目录下新建 com.struts2.model 包，在该包下新建 Book 类，代码如下。

```
package com.struts2.model;
import java.io.Serializable;
public class Book implements Serializable {
    private String title;//书名
    private  float price;//价格
    private int amount;//购买数量
    /*下面是上面 3 个属性的 set、get 方法，这里省略*/
}
```

(3)　编写 BookAction 类。BookAction 是一个 Action 类，该类只是提供了一个保存用户提交的图书信息的 Map 对象。在 src 目录下新建 com.struts2.actions 包，在该包下新建 BookAction 类，继承自 ActionSupport，完整的代码如下所示。

```
package com.struts2.actions;
import java.util.Map;
import com.opensymphony.xwork2.ActionSupport;
public class BookAction extends ActionSupport {
    //保存 Book 对象的 Map
    private Map books;
    @Override
    public String execute() throws Exception {
        return SUCCESS;
    }
    public Map getBooks() {
        return books;
    }
    public void setBooks(Map books) {
        this.books = books;
    }
}
```



books 实例变量的类型是 Map 接口，并且我们没有对它进行初始化。

在 BookAction 类中，没有给出 books Map 的 key 和 value 的类型，Map 的 key 和 value 的类型可以在 BookAction-conversion.properties 文件中指出。

(4)　继续在 com.struts2.actions 包下，即 BookAction 类的同目录下新建 BookAction-conversion.properties 文件，代码内容如下。

```
#Map 类型的属性 books 的 key 类型为 String
Key_books=java.lang.String
#Map 类型的属性 books 的 value 类型为 Book
Element_books=com.struts2.model.Book
```

(5) 在 WebRoot 下新建 addBook.jsp 页面，用于收集订购图书信息。代码如下。

```
    <%@ page language="java" import="java.util.*" pageEncoding="GBK"%>
    <%@taglib prefix="s" uri="/struts-tags" %>
    <s:form action="convert/bookConfirm.action" method="post" theme="simple">
        <TABLE cellSpacing=0 cellPadding=2 width="95%" align=center border=0>
            <TR>
                <TD>书名</TD>
                <TD >价格</TD>
                <TD>数量</TD>
            </TR>
        <s:iterator value="new int[3]" status="status">
            <TR>
            <s:set name="index" value="#status.index+1"/>
            <%--
            <s:set name="books" value='"books["+"\'book"+#index+"\']"'/>
            --%>
            <s:set name="books" value="'books.'+'book'+#index"/>
                <TD ><s:textfield name="%{#books+'.title'}"/></TD>
                <TD><s:textfield name="%{#books+'.price'}"/></TD>
                <TD><s:textfield name="%{#books+'.amount'}"/></TD>
            </TR>
        </s:iterator>
            <TR>
                <TD colspan="3" align="center">
                    <input type="submit" value="订购图书">
                </TD>
            </TR>
        </TABLE>
    </s:form>
```

如果是用 Map 来保存数据，那么在为表单输入元素命名时，需要使用 books[‘key’]或者 books.key 格式的名字，在上面的代码中，同时给出了这两种格式的命名，对于 books[‘key’]这种格式是在 JSP 注释中给出的。

(6) 在 struts.xml 文件中配置 BookAction 类。完整代码如下。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- 指定拦截器的 DTD 信息 -->
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <!-- 通过常量配置 Struts2 所使用的解码集 -->
    <constant name="struts.i18n.encoding" value="gbk" />
```

```
    <constant name="struts.devMode" value="true" />
    <package name="convert" namespace="/convert" extends="struts-default">
        <action name="bookConfirm" class="com.struts2.actions.BookAction">
            <result name="success">/showBooks.jsp</result>
        </action>
    </package>
</struts>
```

(7) 在 WebRoot 下新建获取订购图书信息页面 showBooks.jsp，代码如下。

```
    <%@ page language="java" import="java.util.*" pageEncoding="GBK"%>
    <%@taglib prefix="s" uri="/struts-tags" %>
    <TABLE cellSpacing=0 cellPadding=2 width="95%" align=center border=0>
        <TR>
            <TD>书名</TD>
            <TD >价格</TD>
            <TD>数量</TD>
        </TR>
    <s:iterator value="books">
        <TR>
            <TD ><s:property value="value.title"/></TD>
            <TD><s:property value="value.price"/></TD>
            <TD><s:property value="value.amount"/></TD>
        </TR>
    </s:iterator>
    </TABLE>
```

books 的类型为 Map，使用 iterator 标签迭代 Map 时，实际迭代的是 Map 的 entrySet()方法返回的 Set 集合，在该集合中是一组 Map.Entry 对象。表达式 value 获取 Map 中的值，即 Book 对象，表达式 value.title 则获取 Book 对象的 title 属性。

3.2.4　运行结果

运行 addBook.jsp 页面，分别输入三本书的书名、价格和数量，如图 3-4 所示。



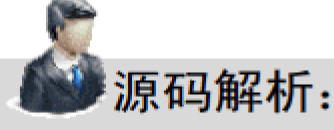
图 3-4　订购图书页面

单击界面中的“订购图书”按钮，表单提交至收集订购图书信息界面 showBooks.jsp，如图 3-5 所示。



图 3-5　收集订购图书信息界面

3.2.5　实例分析

**源码解析：**

在上述案例的 addBook.jsp 页面中，OGNL 表达式“#status.index+1”是为了让索引从 1 开始，表达式“‘books.’+‘book’ +#index”构建 books.book1 格式的字符串，文本输入控件最终的名字格式为：books.book1.title、books.book2.title，以此类推。如果采用 JSP 注释中的代码，则文本输入控件最终的名字格式为：books[‘book’].title…，以此类推。

3.3　自定义类型转换器

如果 Struts 2 内置的类型转换器不能满足应用需求，还可以开发自定义的类型转换器，以便于在不同的需求下使用不同的解决方案。

视频教学：光盘/videos/03/self-defined.avi　　　　长度：15 分钟

3.3.1　基础知识——编写自定义类型转换器

Struts 2 已经实现了一些常用的类型转换，但是毕竟这些类型转换器还是有限制的。如果开发者自己定义数据类型，就必须自定义类型转换器来进行转换。

1．基本类型转换器

要创建一个类型转换器，需要实现 ognl.TypeConverter 接口，该接口中只有一个方法，如下所示。

```
public abstract interface TypeConverter
{
    public abstract Object convertValue(Map<String, Object> paramMap, Object
    paramObject1, Member paramMember, String paramString, Object paramObject2,
    Class paramClass);
}
```

由于该方法过于复杂，所以在 OGNL 中还提供了一个工具类 ognl.DefaultTypeConverter，该类实现了 TypeConverter 接口，并提供了一个简单的 converValue()方法，如下所示。

```
public class DefaultTypeConverter implements
com.opensymphony.xwork2.conversion.TypeConverter
{
    public Object convertValue(Map<String, Object> context, Object value, Class
toType) {
        return convertValue(value, toType);
    }
}
```

其中，convertValue()方法负责完成类型的双向转换，为了实现双向转换，通过判断 toType 的类型即可判断转换的方向，toType 类型是需要转换的目标类型，如：当 toType 类型是 User 类型时，表明需要将字符串转换成 User 实例；当 toType 类型是 String 类型时，表明需要把 User 实例转换成字符串类型。toType 参数和转换方向之间的关系如图 3-6 所示。

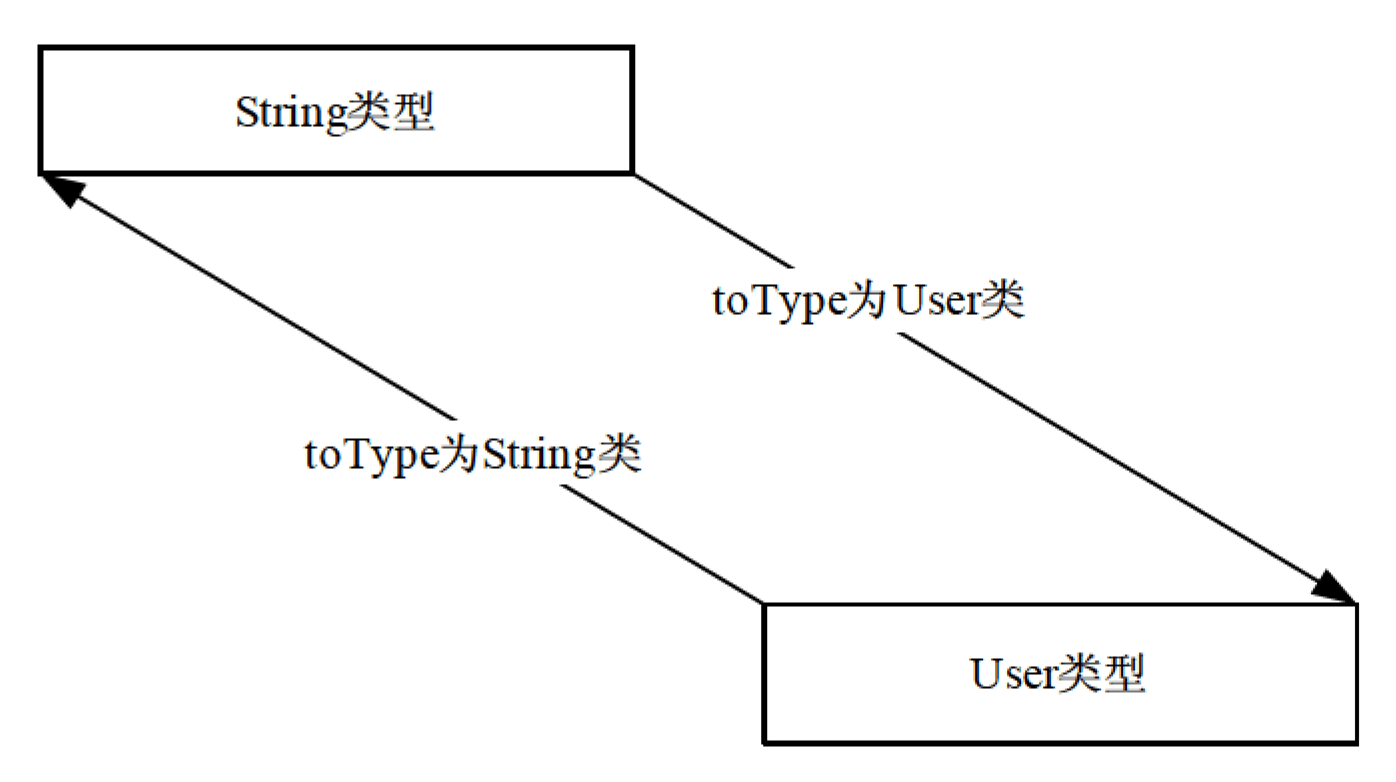


图 3-6　toType 参数和转换方向之间的关系

通过 toType 类型判断了类型转换的方向后，就可以分别实现两个方向的转换逻辑了。实现类型转换器的关键就是实现了 convertValue()方法，该方法有以下三个参数。

- 第一个参数：context 是类型转换环境的上下文。

- 第二个参数：value 是需要转换的参数。随着转换方向的不同，value 参数的值也是不一样的，当把字符串类型向 User 类型转换时，value 是原始字符串数组；当需要把 User 类型向字符串类型转换时，value 是 User 实例。
- 第三个参数：toType 是转换后的目标类型，这个参数前面已经介绍了，这里不赘述。

convertValue()方法的返回值就是类型转换后的值，该值的类型也会随着转换方向的不同而不同，当把字符串向 User 类型转换时，返回值类型就是 User 类型；当需要把 User 类型向字符串类型转换时，返回值类型就是字符串类型。

由此可见，转换器的 convertValue()方法接收需要转换的值，需要转换的目标类型为参数，然后返回转换后的目标值。

2. 基于 Struts 2 的类型转换器

Struts 2 提供了一个 StrutsTypeConverter 的抽象类，这个抽象类是 DefaultTypeConverter 类的子类。开发者可以直接继承这个类来进行类型转换器的构造。通过继承该类来构建类型转换器，可以不用对转换的类型进行判断。StrutsTypeConverter 类的代码如下所示。

```
public abstract class StrutsTypeConverter extends DefaultTypeConverter
{
    //重写 DefaultTypeConverter 类的 convertValue()方法
    public Object convertValue(Map context, Object o, Class toClass)
    {
        if (toClass.equals(String.class))
            return convertToString(context, o);
        if ((o instanceof String[]))
            return convertFromString(context, (String[])(String[])o,
toClass);
        if ((o instanceof String)) {
            return convertFromString(context, new String[] { (String)o },
toClass);
        }
    return performFallbackConversion(context, o, toClass);
    }
    protected Object performFallbackConversion(Map context, Object o, Class
toClass)
    {
        return super.convertValue(context, o, toClass);
    }
    public abstract Object convertFromString(Map paramMap, String[]
paramArrayOfString, Class paramClass);
    public abstract String convertToString(Map paramMap, Object paramObject);
}
```

StrutsTypeConverter 类已经实现了 DefaultTypeConverter 的 convertValue()方法，实现该方法时，它将两个不同的转换方向替换成不同的方法——当需要把字符串转换成复合类型时，调用 convertFromString()抽象方法；当需要把复合类型转换成字符串时，调用 convertToString()抽象方法。图 3-7 展示了对应关系。

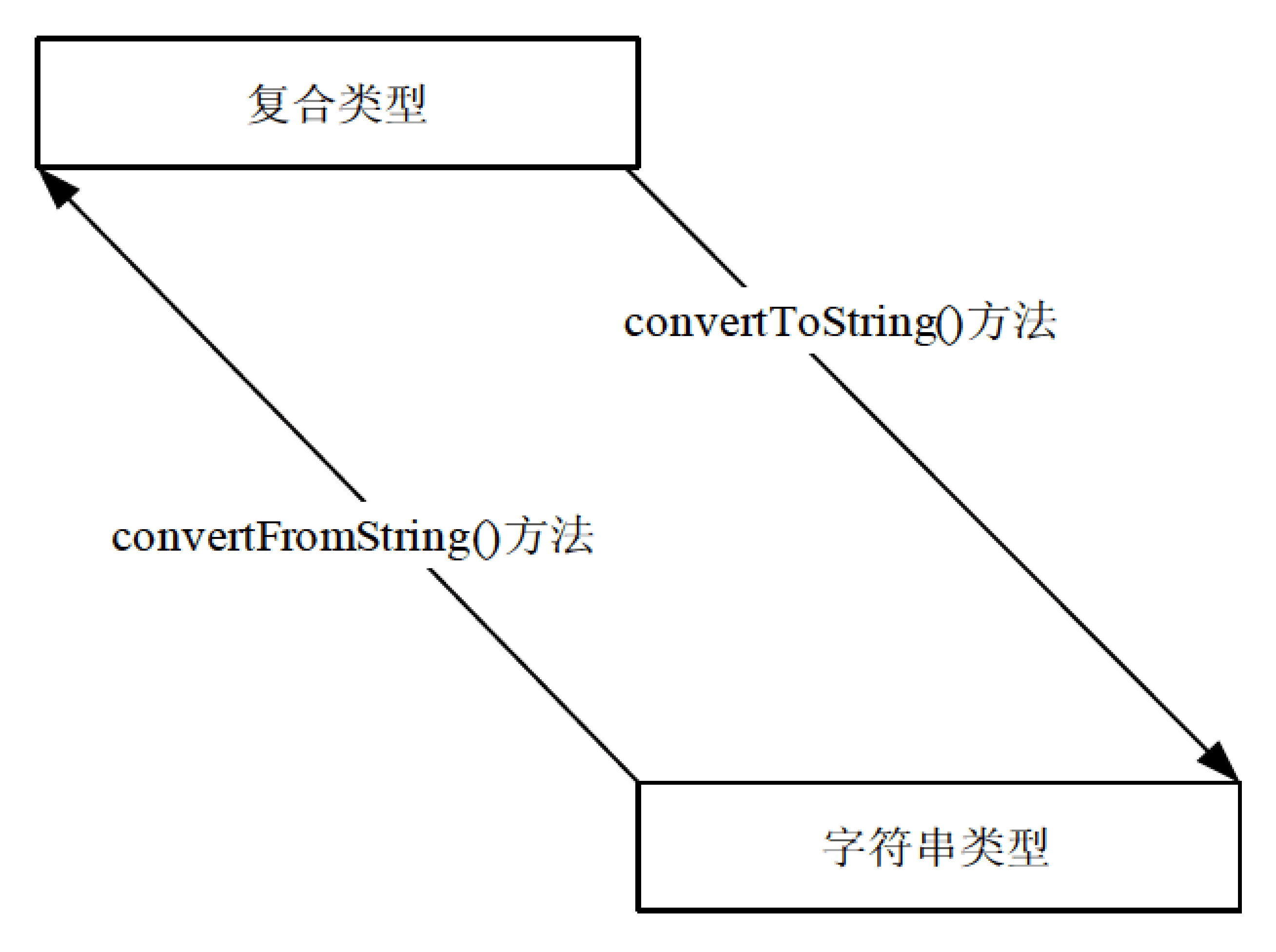


图 3-7　转换方向和方向之间的对应关系

实现了自定义的类型转换器之后，要将类型转换器注册到 Web 应用中，Struts 2 框架才可以正常使用该类型转换器。类型转换器的注册方式有三种：注册局部类型转换器、注册全局类型转换器和使用注解方式注册类型转换器，这三种方式在前面一节中已经详细讲解过，这里不再赘述。

3.3.2　实例描述

上小学的时候老师曾经说过一个圆在平面上是有坐标的，这个坐标用来决定圆在平面上的位置；老师也曾经讲过一个圆的周长要怎么计算，面积要怎么计算。而计算圆的周长和面积需要知道哪些条件？即知道圆的半径就可以计算出圆的周长和面积。

下面这个案例就帮我们完成了一个计算圆的周长和面积的功能。

3.3.3　实例应用

【例 3-3】 计算圆的周长和面积。

(1)　在项目中新建 com.struts2.model 包，并在该包下新建 Point 类，收集圆的坐标信息。内容如下。

```
package com.struts2.model;
public class Point {
    private double x;//横坐标
    private double y;//纵坐标
    /*下面是上面两个属性的 set、get 方法，这里省略*/
}
```

(2) 在 com.struts2.model 包下新建 Circle 类，在该类中定义圆的属性，包括圆的半径和坐标，则需要在其内定义一个 Point 对象属性 center。内容如下。

```
package com.struts2.model;
public class Circle {
    private Double radius;//圆的半径
    private Point center;//坐标类 Point 对象属性
    /*下面是上面两个属性的 set、get 方法，这里省略*/
    …
}
```

(3) 新建 com.struts2.converter 包，并在该包下新建自定义类型转换器类 CircleConverter.java。该类继承自 org.apache.struts2.util.StrutsTypeConverter 类，并重写了父类的 convertFromString(Map context, String[] values, Class toClass)方法和 convertToString(Map context, Object value)方法，实现了类型转换。完整代码如下。

```
package com.struts2.converter;
import java.util.Map;
import org.apache.struts2.util.StrutsTypeConverter;
import com.struts2.model.Circle;
import com.struts2.model.Point;
public class CircleConverter extends StrutsTypeConverter {
    @Override
   public Object convertFromString(Map context, String[] values, Class toClass)
{
        Circle circle=new Circle();
        Point center=new Point();
        String centerValues[]=values[0].split(",");
        //获取圆的横坐标
        double x=Double.parseDouble(centerValues[0]);
        //获取圆的纵坐标
        double y=Double.parseDouble(centerValues[1]);
        //给 Point 坐标类赋值
        center.setX(x);
        center.setY(y);
        //获取圆的半径，并直接赋值给 Circle 类中的 radius 属性
        circle.setRadius(Double.parseDouble(centerValues[2]));
        //将已经赋值的 Point 类对象 center 赋给 Circle 类中的 Point 对象属性 center
        circle.setCenter(center);
        return circle;
    }
    @Override
    public String convertToString(Map context, Object value) {
        Circle circle=(Circle)value;
        return circle.toString();
    }
}
```

(4)　在 com.struts2.actions 包下新建 CircleAction 类，获取圆的信息并保存。代码如下。

```
package com.struts2.actions;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
import com.struts2.model.Circle;
public class CircleAction extends ActionSupport {
    private Circle circle;
    @Override
    public String execute() throws Exception {
        //计算圆的周长
        double c=2*3.14*circle.getRadius();
        //计算圆的面积
        double s=3.14*circle.getRadius()*circle.getRadius();
        //保存圆的信息
        ActionContext.getContext().put("message",圆心坐标:
("+circle.getCenter().getX()+","+circle.getCenter().getY()+")----- 圆的半径:
"+circle.getRadius()+"  -----  圆的周长: "+c+" ----- 圆的面积"+s);
        return SUCCESS;
    }
    public Circle getCircle() {
        return circle;
    }
    public void setCircle(Circle circle) {
        this.circle = circle;
    }
}
```

(5)　在 com.struts2.actions 包下新建 CircleAction-conversion.properties 文件，将类型转换器注册到 Web 应用程序中。该文件只有一行代码，如下所示。

```
circle=com.struts2.converter.CircleConverter
```

(6)　在 WebRoot 下新建 createCircle.jsp 页面，可以输入圆的横坐标、纵坐标和半径的一个表单页面。关键代码如下。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@taglib prefix="s" uri="/struts-tags" %>
请分别输入圆的横坐标、纵坐标、半径
    <s:form action="convert/circle.action" method="post">
        <table align="center" border="0"  width="400">
            <tr ><td colspan="2"><font color="#3737DE"></font></td></tr>
            <tr>
                <td><input type="text" name="circle"/></td>
                <td><font color="red">x,y,radius</font></td>
            </tr>
            <tr><td colspan="3" align="center"><s:submit
value="submit"/></td></tr>
        </table>
    </s:form>
```

(7) 创建 circleMessage.jsp 页面，获取圆的信息并输出。关键代码如下。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@taglib prefix="s" uri="/struts-tags" %>
<s:property value="message"/>
```

上面代码的<s:property>标签中的 value 值，是在 CircleAction 类的 execute()方法中保存在 ActionContext 中的 message 值，故在这里直接调用“message”就可得到 Value 的值。

(8) 在 struts.xml 文件中配置 CircleAction 类，代码如下。

```
    <package name="convert" namespace="/convert" extends="struts-default">
        <action name="circle" class="com.struts2.actions.CircleAction">
            <result name="success">/circleMessage.jsp</result>
        </action>
    </package>
```

3.3.4 运行结果

运行 createCircle.jsp 页面，在文本框中输入“100,200,300”，如图 3-8 所示。

单击 submit 按钮，提交至 circleMessage.jsp 页面，输出圆的信息，如图 3-9 所示。




图 3-8 定义圆界面



图 3-9 输出圆的信息界面

3.3.5 实例分析



源码解析：

在上述例子的 CircleConverter 转换器类中，重写了 org.apache.struts2.util.StrutsTypeConverter 类的 convertFromString(Map context, String[] values, Class toClass)方法，在该方法中获取到了表单元素中以逗号隔开的三个值：圆的横坐标、圆的纵坐标和圆的半径，并分别赋给了 Point 类中的 x 属性、y 属性和 Circle 类中的 radius 属性，从而在 CircleAction 中调用 Circle 中的属性和 Point 中的属性即可计算出圆的周长和面积。

3.4　类型转换中的异常处理

在 JSP 文件中提供用户输入信息，而偶然或者恶意的输入错误，都会导致程序出现异常。因此，必须对用户输入的数据进行校验，例如日期校验、整数校验和字符串长度校验等。本节要介绍的是数据在类型转换时的校验。例如出生日期必须是日期类型，但用户却输入非日期类型字符串数据，这时就需要进行数据类型转换的异常处理。

视频教学：光盘/videos/03/build-in.avi　　　长度：11 分钟

3.4.1　基础知识——类型转换中的异常处理

表现层数据是由用户输入的，用户输入的信息是非常复杂的，用户的偶然错误或者是 Cracker(破坏者)的恶意输入，都可能导致系统出现非正常的情况。实际上，表现层数据涉及的数据校验和类型转换两个处理是紧密相关的，只有当输入数据是有效数据时，系统才可以进行有效的类型转换。当然，有时候即使用户输入的数据能进行有效转换，但依然是非法数据(假设需要输入一个人的年龄，输入 200 则肯定是非法数据)。因此，进行有效的类型转换是基础，只有当数据完成了有效的类型转换后，下一步才去做数据校验。

1．类型转换异常拦截器

Struts 2 提供类型转换异常处理机制，它提供名称为 conversionError 的拦截器，这个拦截器被注册在默认拦截器栈中。如果 Struts 2 在类型转换过程中出现异常，那么该拦截器就会进行拦截，并将异常信息封装成一个 fieldError，然后在视图页面上显示出来。



Struts 2 提供类型转换异常处理机制，整个过程无需开发者参与，Struts 2 的类型转换器和 conversionError 拦截器会自动实现。

查看 Struts 2 框架的默认配置文件 struts-default.xml，该文件中有如下配置片段。

```
<interceptor name="conversionError"
class="org.apache.struts2.interceptor.StrutsConversionErrorInterceptor"/>
<interceptor-stack name="defaultStack">
<!--省略其他拦截器引用-->
…
<!--处理类型转换错误的拦截器-->
<interceptor-ref name="conversionError"/>
<!--处理数据校验的拦截器-->
<interceptor-ref name="validation">
        <param name="excludeMethods">input,back,cancel,browse</param>
</interceptor-ref>
<!--省略其他拦截器-->
…
</interceptor-stack>
```

从上面的配置中可以看出，在 struts-default.xml 文件中定义了 conversionError 拦截器，类型是 org.apache.struts2.interceptor.StrutsConversionErrorInterceptor，这个拦截器已被包含在 defaultStack 拦截器栈中。



org.apache.struts2.interceptor.StrutsConversionErrorInterceptor 拦截器继承自 ConversionErrorInterceptor 类，它只在字段值不是 null、不是 " " 或者不是{ " " }(表示只有一个空字符串元素的字符串数组)的情况下，才会把转换错误从 ActionContext 添加到 Action 的字段错误中。这在 Web 环境下是有用的。

如果 Struts 2 类型转换器执行类型转换时出现错误，上述 conversionError 拦截器负责将对应错误封装成表单域错误(fieldError)，并将这些错误信息放入 ActionContext 中。图 3-10 显示了 Struts 2 类型转换中的错误处理流程。

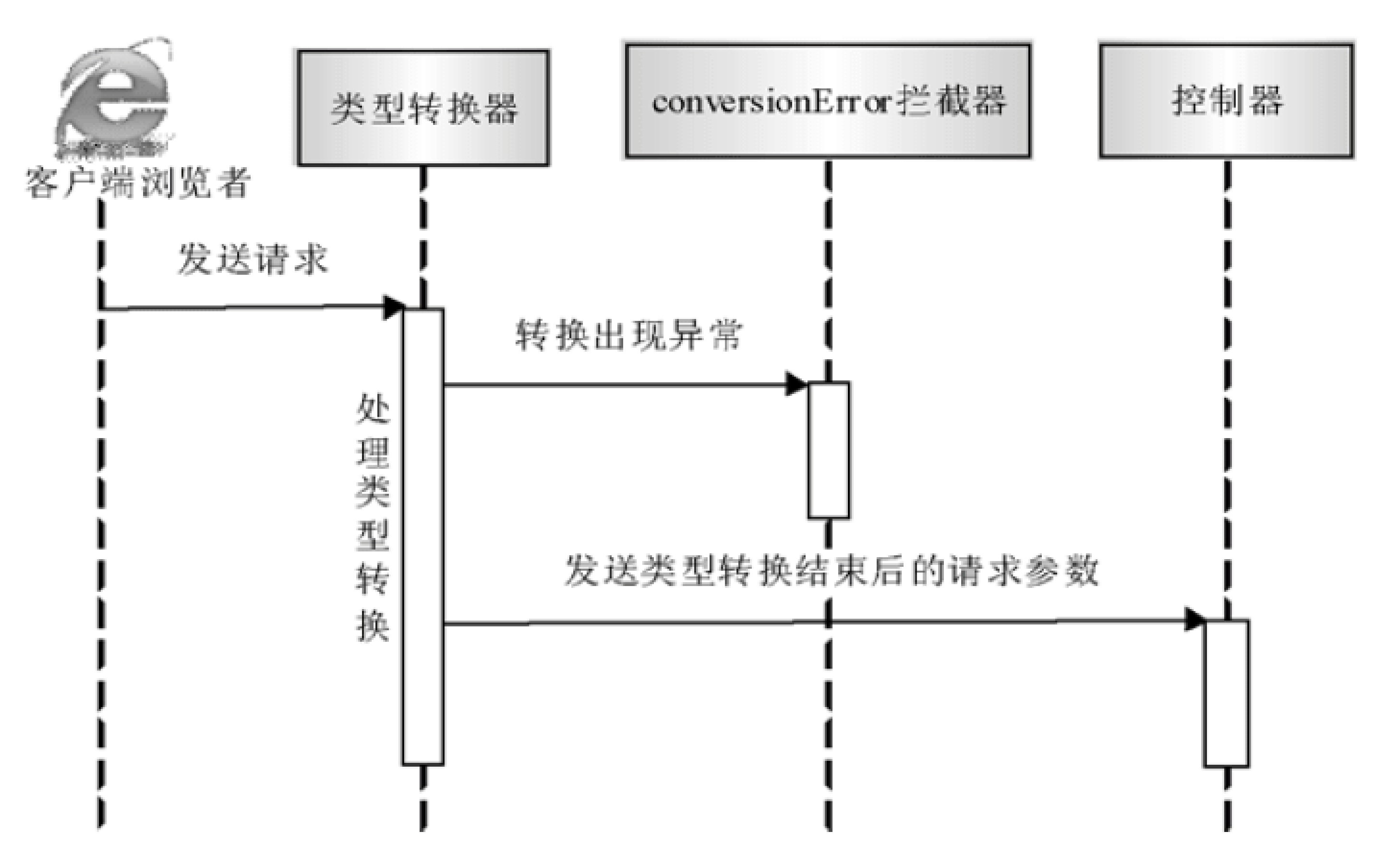


图 3-10　Struts 2 类型转换的错误处理流程

在图 3-10 中只显示了类型转换器、conversionError 拦截器和控制器之间的顺序图，并未完全刻画出系统中的其他成员。当 conversionError 拦截器对转换异常进行处理后，系统会跳转到名为 input 的逻辑视图。



必须指出的是，为了让 Struts 2 框架处理类型转换的错误，以及使用后面的数据校验机制，系统的 Action 类都应该通过继承 ActionSupport 类来实现。ActionSupport 类为完成类型转换错误处理，数据校验实现了许多基础工作。

2. 处理类型转换错误

下面将以一个最简单的局部类型转换器为例，介绍如何处理类型转换的错误。

重新创建一个 Action 类，让系统的 Action 类继承 Struts 2 的 ActionSupport 类。内容如下。

```
//为了正常使用系统的类型转换错误处理机制，让 Action 类继承 ActionSupport 类
public class UserAction extends ActionSupport
{
    private User user;
```

```
    private String tip;
    /*下面是上面两个属性的 set、get 方法，这里省略*/
    //省略 execute()方法
    …
}
```

当类型转换异常时，conversionError 拦截器会处理该异常，然后转入名为 input 的逻辑视图，因此应该为该 Action 增加名为 input 的逻辑视图定义。在 struts.xml 文件中配置如下。

```
<struts>
    <!-- 配置 Struts 2 的包空间 -->
    <package name="user" extends="struts-default">
        <!-- 定义处理用户请求的 Action -->
        <action name="user" class="com.struts2.actions.UserAction">
            <!-- 配置名为 input 的逻辑视图，当校验失败后转入该逻辑视图 -->
            <result name="input">/input.jsp</result>
            <!-- 配置名为 success 的逻辑视图 -->
            <result name="success">/success.jsp</result>
        </action>
    </package>
</struts>
```

经过上面的配置，如果用户输入信息不能成功转换成用户实例，系统将转入 input.jsp 页面，等待用户再次输入。

3．输出类型转换错误

当类型转换处理失败后，系统将进入名为 input 的逻辑视图，但在该页面没有任何提示，这样让用户会感到疑惑。

前面已经讲述过，conversionError 会负责将转换错误封装成 fieldError，并将其放在 ActionContext 中。为了在 input 视图对应的页面中输入转换错误，只需要在页面中使用<s:fielderror/>标签即可输出该类型转换错误信息。图 3-11 显示了类型转换出错信息的页面。

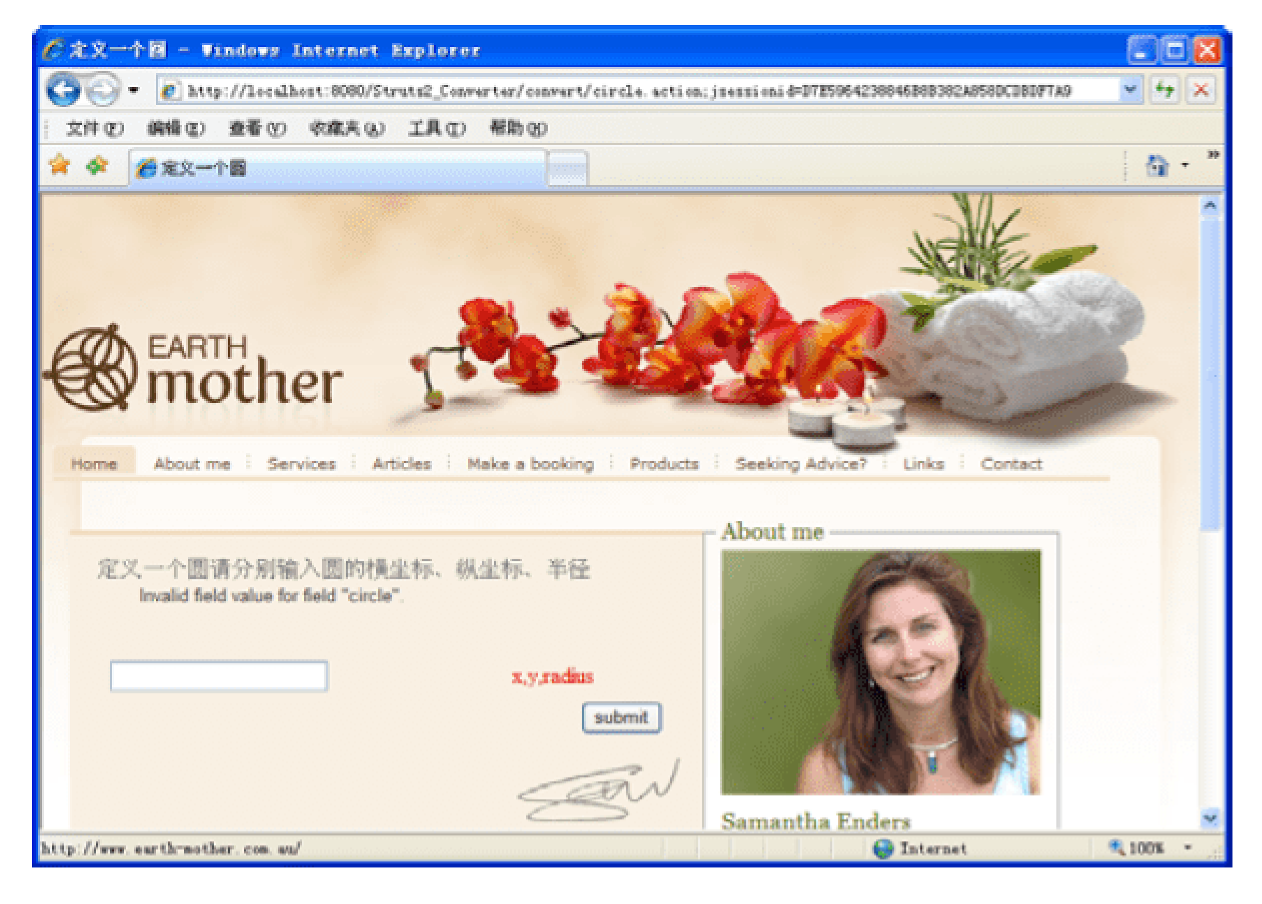


图 3-11　显示类型转换错误的界面

正如图 3-11 所显示的，当某个 Field 类型转换失败后，将出现“Invalid field value for field xxx”的错误信息，其中“xxx”是 Action 中的属性名，也是该属性对应的请求参数的名。对于中文环境而言，通常希望看到中文的提示信息，因此应该在应用中改变该提示信息。Struts 2 允许改变类型转换失败后的错误提示信息，只要在应用的国际化资源文件中增加如下一行代码即可。

```
#改变默认的类型转换失败后的错误提示信息
xwork.default.invalid.fieldvalue={0}字段类型转换失败!
```



由于上面资源文件中包含了非西欧字符，因此必须使用 native2ascii 命令来处理该文件的非西欧字符。

上面资源项中的 xwork.default.invalid.fieldvalue.key 改变提示信息的 key，而后面的信息则是用户希望在页面中显示的提示信息。某些时候，可能还需要对特定字段指定特定的提示信息，此时可以提供该 Action 的局部资源文件，局部资源文件的文件名命名如下。

```
ClassName.properties
```

在该文件中增加如下一项。

```
#为某个 Action 特定属性指定特定的转换失败提示信息
invalid.fieldvalue.属性名=提示信息
```

其中“invalid.fieldvalue”部分是固定的，而“属性名”是 Action 中的属性名，也是请求参数名，该项的 value 就是指定的类型转换失败提示信息，该信息在页面中的显示可以自定义 CSS 样式来设置。

3.4.2 实例描述

一个公司的内部管理系统中，一般都会有用户管理这个模块，这个模块可用于将刚进入公司的人注册成为管理系统的用户。有时候因为公司业务繁忙，需要招进一大批的人才，此时一条一条地添加新招人才信息是很麻烦的事。30 条信息还可以应付，对于一个大公司来说，招聘 80 个人也是很有可能的，那么也要一条一条地添加这 80 个人的信息吗？这是不是意味着管理员还需要是一个身强体壮的人呢？答案是不需要的，我们可以一次性添加多条信息。

下面这个例子就为公司内部管理系统解决了这个信息繁多的问题！

3.4.3 实例应用

【例 3-4】 采用集合转换错误处理实现多个用户同时注册功能。

(1) 在 com.struts2.model 包下新建实体类 User.java，它有三个属性：username、age 和 birth，分别表示用户的用户名、年龄和出生日期。代码如下。

```
package com.struts2.model;
import java.util.Date;
/**
```

```
 * 用户类
 * @author Administrator
 *
 */
public class User {
    private String username;//用户名
    private int age;//年龄
    private Date birth;//出生日期
    /*下面是上面三个属性的 set、get 方法，这里省略*/
}
```

(2) 在 com.struts2.actions 包下新建 UserAction 类，它有一个集合属性 user，并定义了该集合元素类型为 user。UserAction 类的内容如下。

```
package com.struts2.actions;
import java.util.List;
import com.opensymphony.xwork2.ActionSupport;
import com.struts2.model.User;
public class UserAction extends ActionSupport {
    //以一个 List 属性来封装一个字符串请求参数
    private List<User> user;
    /**
     * 处理用户请求的 execute()方法，直接返回 success 字符串
     */
    public String execute() throws Exception {
        return SUCCESS;
    }
    public List<User> getUser() {
        return user;
    }
    public void setUser(List<User> user) {
        this.user = user;
    }
}
```

(3) 在 com.struts2.converter 包下新建转换器类——UserConverter.java，内容如下。

```
package com.struts2.converter;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import org.apache.struts2.util.StrutsTypeConverter;
import com.struts2.model.User;
/**
 * 用户注册类型转换器
 * @author Administrator
 *
```

```
 */
public class UserConverter extends StrutsTypeConverter {
    //实现从 String 类型转换成复合类型的方法
   public Object convertFromString(Map context, String[] values, Class toClass)
{
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");
        //如果添加了多个用户的信息
        if (values.length > 1)
        {
            //定义一个 List 集合
            List<User> result = new ArrayList<User>();
            //循环遍历多个用户的信息
            for (int i = 0; i < values.length; i++)
            {
                User user = new User();
                //把表单的每个元素的值以逗号分开，并组成一个数组
                String[] userValues = values[i].split(",");
                //把相应的表单元素值赋值给 User 类中的属性
                user.setUsername(userValues[0]);
                user.setAge(Integer.parseInt(userValues[1]));
                try {
                    user.setBirth(sdf.parse(values[2]));
                } catch (ParseException e) {
                    e.printStackTrace();
                }
                //把赋值后的 User 类添加至集合中
                result.add(user);
            }
            //返回集合
            return result;
        }
        //如果只添加了一个用户的信息，或没有添加一个用户的信息，把表单元素的值赋给 User
类中的属性，并返回 User 类对象
        User user = new User();
        String[] userValues = values[0].split(",");
        user.setUsername(userValues[0]);
        user.setAge(Integer.parseInt(userValues[1]));
        try {
            user.setBirth(sdf.parse(values[2]));
        } catch (ParseException e) {
            e.printStackTrace();
        }
        return user;
    }
    //实现从复合类型转换成 String 类型
    public String convertToString(Map context, Object o) {
        //如果复合类型是 User 类型
```

```
        if ((o instanceof User))
        {
            User user = (User)o;
            return "<用户名：" + user.getUsername() + ",年龄：" + user.getAge()
+ ">";
        }
        //如果复合类型为 List 类型
        if ((o instanceof List))
        {
            List<User> users = (List<User>)o;
            String result = "[";
            for (User user : users)
            {
                result = result + "<用户名：" + user.getUsername() + ",年龄：" +
user.getAge() + ">";
            }
            return result + "]";
        }
        return "";
    }
}
```

(4) 创建了类型转换器类之后，需要把类注册到 Web 应用程序中，在这里使用全局类型转换器注册方式来注册，即在 src 目录下新建 xwork-conversion.properties 文件，内容如下。

```
#所有包含 com.struts2.model.User 类的 Action 都使用
com.struts2.converter.UserConverter 转换器
com.struts2.model.User=com.struts2.converter.UserConverter
```

定义了这样的内容后，当 Web 程序中的任何一个 Action 类中包含了 com.struts2.model.User 对象时，都会使用 com.struts2.converter.UserConverter 转换器来对表单输入数据进行 String 与复合类型的转换。

(5) 创建国际化资源文件，改变类型转换失败信息。在 src 目录下新建 globalMessages.properties 文件，该文件是英文环境下要使用的国际化资源文件。内容如下。

```
#改变默认的类型转换失败后的提示信息
xwork.default.invalid.fieldvalue={0}invalid field value for field!
```

(6) 接着在 src 目录下新建中文环境下的国际化资源文件(globalMessages_zh_CN.properties)，内容如下。

```
xwork.default.invalid.fieldvalue= {0}字段类型转换失败!
```

(7) 在 struts.xml 文件中配置程序的国际化资源文件和 UserAction 类，配置如下。

```
<!-- 通过常量配置 Struts2 所使用的解码集 -->
<constant name="struts.i18n.encoding" value="gbk" />
<constant name="struts.devMode" value="true" />
<constant name="struts.custom.i18n.resources" value="globalMessages"/>
<package name="convert" namespace="/convert" extends="struts-default">
```

```
    <action name="user" class="com.struts2.actions.UserAction">
            <result name="input">/inputUser.jsp</result>
            <result name="success">/index.jsp</result>
    </action>
</package>
```



当 conversionError 拦截器拦截到类型转换失败后，系统将跳转至“input”逻辑视图，故这里配置的“input”的 Result 不可少。

(8) 在 WebRoot 下新建 inputUser.jsp 页面，用于录入用户信息，每次最多可注册三个用户的信息，代码如下。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@taglib prefix="s" uri="/struts-tags" %>
<%--类型转换失败提示信息--%>
<s:fielderror />
<form action="convert/user.action" method="post">
    <table align="center" width="360">
        <tr align="center">
            <td style="font-size: 12px;">请分别输入三个用户的用户名、年龄、出生日期，
以逗号隔开</td>
        </tr>
    <s:iterator value="new int[3]" status="stat">
        <tr>
            <td>
                第<s:property value="%{#stat.index+1}"/>个用户信息
                <input name='user' type="text"/>
            </td>
        </tr>
    </s:iterator>
        <tr align="center">
            <td><input type="submit" value="确定"/><input type="reset" value="
重设"/></td>
        </tr>
    </table>
<form>
```

(9) 在 WebRoot 下新建用户注册成功页面 success.jsp，并在页面中输出新注册的用户信息，使用<s:property …/>标签即可输出用户信息。

3.4.4 运行结果

运行 inputUser.jsp 页面，当输入的信息不符合“String,int,Date”格式的类型数据时(如图 3-12 所示)，提交表单后还是跳转到这个页面，并提示类型转换失败信息，如图 3-13 所示。

3.4.5　实例分析



源码解析：

在上面的例子中，inputUser.jsp 页面定义了三个 user 的请求参数，但是服务器只会接受一个名为 user 的请求参数，该请求参数的值是一个字符串数组。在 UserAction 类中使用了泛型来限制集合元素的类型，因此系统的类型转换器会作用于该 Action 的 user 属性，如果用户的输入不能正确转换成对应的类型，则 conversionError 拦截器会起作用，故当类型转换失败后会提示如图 3-13 所示的错误信息。



图 3-12　录入无效信息



图 3-13　类型转换失败提示信息

3.5　使用类型转换注解

作为 ClassName-conversion.properties 文件的替代，可以使用 Struts 2 提供的类型转换注解来配置类型转换器。这一节将介绍 Struts 2 中使用类型转换注解的相关知识。

**视频教学：**光盘/videos/03/zhujie.avi　　　　长度：15 分钟

3.5.1　基础知识——使用类型转换注解

Struts 2 中用于类型转换的注解共有 6 个，如下所示。

- TypeConversion
- Conversion
- Element
- Key
- KeyProperty
- CreateIfNull

1．TypeConversion 注解

TypeConversion 注解应用于属性和方法级别，它的参数如表 3-1 所示。

表 3-1　TypeConversion 注解的参数

参　数	类　型	是否必需	默 认 值	描　述
key	String	否	被标注的属性名	指定要转换的属性名或类名，依赖于 type 参数的值
type	ConversionType	否	ConversionType.CLASS	ConversionType 的枚举值，可以是 APPLICATION 或者 CLASS
rule	ConversionRule	否	ConversionRule. PROPERTY	ConversionRule 的枚举值，可以是 PROPERTY、KEY、KEY_PROPERTY、ELEMENT、COLLECTION(已被弃)或者 MAP
converter	String	和 value 参数任选其一	无	指定用作转换器的 TypeConverter 实现类的类名。这个参数不能和 Conversion Rule.KEY_PROPERTY 一起使用
value	String	和 converter 参数任选其一	无	和 ConversionRule_PROPERTY 一起使用时，指定一个值。如果使用 Conversion Type.APPLICATION，则不能使用这个参数

TypeConversion 注解可以指定类范围转换或者应用程序范围转换的规则。

1)　类范围转换

设置 type 为：type=ConversionType.CLASS，转换规则将从一个名为 ClassName-conversion. properties 文件中转配，该文件和其相关的 Action 类在同一个包中。

2)　应用程序范围转换

设置 type 为：type=ConversionType.APPLICATION，转换规则将从 xwork-conversion.properties 文件中装配，该文件位于 CLASSPATH 的根路径下。

下面使用 TypeConversion 注解来配置用户注册程序中的日期类型转换器，编辑 User.java，在 setBirth()方法上使用 TypeConversion 注解，代码如下所示。

```
...
//应用程序范围转换
@TypeConversion(
```

```
    type=ConversionType.APPLICATION,
    key="java.util.Date",
    converter="com.struts2.converter.DateTypeConverter"
)
//应用类范围转换
@TypeConversion(
    type=ConversionType.CLASS,//可以省略
    key="birth",//可以省略
    converter="com.struts2.converter.DateTypeConverter"
)
public void setBirth(Date birth) {
    this.birth = birth;
}
…
```

2．Conversion 注解

Conversion 注解可以让类型转换应用到类型(TYPE)级别，即可以应用到类、接口(包括注解类型)或枚举声明。Conversion 注解只有一个参数，如表 3-2 所示。

表 3-2　Conversion 注解的参数

参　数	类　型	是否必需	默 认 值	描　述
conversions	TypeConversion[]	否	无	允许类型转换被应用到类型(TYPE)级别

在 User 类上使用 Conversion 注解来配置日期类型转换器，代码如下。

```
//使用 Conversion 注解
@Conversion(
    //指定 conversions 参数
    conversions=
    {
        //使用 TypeConversion 注解
        @TypeConversion(
            type=ConversionType.CLASS,
            key="birth",
            converter="com.struts2.converter.DateTypeConverter"
        )
    }
)
public class User implements Serializable
…
```

3．Element 注解

Element 注解用于指定 Collection 或 Map 中的元素类型，该注解只能用于字段或方法级别。Element 注解只有一个参数，如表 3-3 所示。

表 3-3　Element 注解的参数

参　数	类　型	是否必需	默 认 值	描　述
value	Class	否	java.lang.Object.class	指定 Collection 或 Map 中的元素类型

Element 注解用于替代 ClassName-conversion.properties 文件中的 Element_xxx 的配置，代码如下。

```
//指定 List 元素类型为 com.struts2.model.User 类
@Element(com.struts2.model.User.class)
private List users;
//指定 Map 元素类型为 com.struts2.model.User 类
@Element(com.struts2.model.User.class)
private Map users;
```



在 J2SE5.0 以后的版本中，可以直接使用泛型技术来指定元素的类型。

4．Key 注解

Key 注解用于指定 Map 中的 key 的类型，该注解只能用于字段或方法级别。Key 注解只有一个参数，如表 3-4 所示。

表 3-4　Key 注解的参数

参　数	类　型	是否必需	默 认 值	描　述
value	Class	否	java.lang.Object.class	指定 Map 中的 key 的类型

Key 注解用于替代 ClassName-conversion.properties 文件中的 Key_xxx 的配置，代码如下所示。

```
//指定 Map 元素类型为 com.struts2.model.User 类
@Element(com.struts2.model.User.class)
//指定 Map 中的 key 类型为 java.lang.String 类型
@Key(java.lang.String.class)
private Map users;
```



在 J2SE5.0 以后的版本中，可以直接使用泛型技术来指定 Map 中的 key 的类型。

5．KeyProperty 注解

KeyProperty 注解指定用于索引集合元素的属性名，该注解只能用于字段或方法级别，KeyProperty 注解只有一个参数，如表 3-5 所示。

表 3-5　KeyProperty 注解的参数

参　数	类　型	是否必需	默 认 值	描　述
value	String	否	id	指定用于索引集合元素的属性名

KeyProperty 注解用于替代 ClassName-conversion.properties 文件中的 KeyProperty_xxx 的配置，代码如下。

```
//指定 Set 集合中的元素类型为 com.struts2.model.User 类
@Element(com.struts2.model.User.class)
//指定 Set 集合索引为 User 类型的 username 属性
@KeyProperty("username")
private Set users=new LinkedHashSet();
```

6. CreateIfNull 注解

CreateIfNull 注解指定在引用的集合元素为 null 时，是否让框架创建它。该注解只能用于字段或方法级别。CreateIfNull 注解只有一个参数，如表 3-6 所示。

表 3-6　CreateIfNull 注解的参数

参　数	类　型	是否必需	默 认 值	描　述
value	boolean	否	false	指定在引用的集合元素为 null 时，是否让框架创建它

CreateIfNull 注解用于替代 ClassName-conversion.properties 文件中的 CreateIfNull_xxx 的配置，代码如下。

```
//指定 Set 集合中的元素类型为 com.struts2.model.User 类
@Element(com.struts2.model.User.class)
//指定 Set 集合中的访问索引为 User 类中的 username 属性名
@KeyProperty("username")
//指定引用 Set 集合元素为 null 时，让框架创建它
@CreateIfNull(true)
private Set users=new LinkedHashSet();
```

3.5.2　实例描述

大家应该都有在不同的系统中注册一个属于自己账号的经验吧！那么当我们在年龄输入框中输入几个英文字母，或是在出生日期的输入框中输入非日期格式的字符，将会出现什么样的结果呢？或许通过前面章节的讲解，这些错误都可以得以解决，那么，如何使用类型转换的注解方式来实现用户注册功能呢？下面来做一下吧！

3.5.3　实例应用

【例 3-5】 运用类型转换注解方式实现用户注册功能。

(1)　在 com.struts2.model 包下新建 User.java 类，它有三个属性，分别是 username、age、birth，并有相应的 set、get 方法。在此直接使用 3.4.4 节案例中的 User.java 类。

(2) 在 com.struts2.converter 包下新建整数类型转换器类——IntegerTypeConverter.java，代码如下。

```
package com.struts2.converter;
import java.util.Map;
import org.apache.struts2.util.StrutsTypeConverter;
import com.opensymphony.xwork2.conversion.TypeConversionException;
/**
 * 整数类型转换器
 * @author Administrator
 *
 */
public class IntegerTypeConverter extends StrutsTypeConverter {

    //把表单元素中输入的字符串类型转换成整数类型
   public Object convertFromString(Map context, String[] values, Class toClass)
   {
        try {
            return Integer.parseInt(values[0]);
        } catch (Exception e) {
            //当解析出现异常时，抛出 TypeConversionException 异常，以通知 Struts 2
发生了转换错误
            throw new TypeConversionException(
                    e.getMessage()+"["+values+"-class:"+toClass+"]"
            );
        }
    }
    //把表单元素中输入的复合类型转换成字符串类型
    public String convertToString(Map context, Object o) {
        return o.toString();
    }
}
```

(3) 在 com.struts2.converter 包下新建日期格式类型转换器类——DateTypeConverter. java，代码如下。

```
package com.struts2.converter;
import java.text.SimpleDateFormat;
import java.util.Map;
import org.apache.struts2.util.StrutsTypeConverter;
import com.opensymphony.xwork2.conversion.TypeConversionException;
public class DateTypeConverter extends StrutsTypeConverter {
    //定义日期格式
    private static SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");

    //把表单元素输入的字符串类型转换成日期格式类型
   public Object convertFromString(Map context, String[] values, Class toClass)
   {
```

```
        try {
            //使用指定的日期格式解析字符串值，返回 Date 对象
            return sdf.parse(values[0]);
        } catch (Exception e) {
            //当解析出现异常时，抛出 TypeConversionException 异常，以通知 Struts 2
发生了转换错误
            throw new TypeConversionException(
                    e.getMessage()+"["+values+"-class:"+toClass+"]"
            );
        }
    }
    //把表单元素输入的复合类型转换成字符串类型
    public String convertToString(Map context, Object o) {
        //使用指定的日期格式化 Date 对象，返回字符串
        return sdf.format(o);
    }
}
```

(4)　修改 User 类，指定 age 属性和 birth 属性要加载的类型转换器。完整代码如下。

```
package com.struts2.model;
import java.io.Serializable;
import java.util.Date;
import com.opensymphony.xwork2.conversion.annotations.TypeConversion;
/**
 * 用户类
 * @author Administrator
 *
 */
public class User implements Serializable {
    private String username;//用户名
    private int age;//年龄

    private Date birth;//出生日期
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public int getAge() {
        return age;
    }
    //应用类范围转换，指定 age 属性的类型转换器是 com.struts2.converter.IntegerType
    Converter 类
    @TypeConversion(
        converter="com.struts2.converter.IntegerTypeConverter"
    )
```

```
    public void setAge(int age) {
        this.age = age;
    }
    public Date getBirth() {
        return birth;
    }
    //应用类范围转换，指定 birth 属性的类型转换器是 com.struts2.converter.DateType
    Converter 类
    @TypeConversion(
        converter="com.struts2.converter.DateTypeConverter"
    )
    public void setBirth(Date birth) {
        this.birth = birth;
    }
}
```

(5) 在 com.struts2.actions 包下新建用户注册 Action 类——RegistAction.java，在它里面引用了 User 动作类。完整代码如下。

```
package com.struts2.actions;
import com.opensymphony.xwork2.ActionSupport;
import com.struts2.model.User;
public class RegistAction extends ActionSupport {
    private User user;//用户
    @Override
    public String execute() throws Exception {
        return SUCCESS;
    }
    /*下面是上面 user 属性的 set、get 方法，这里省略*/
}
```

(6) 在 src 下新建 globalMessages.properties 文件，即英文环境的国际化资源文件，用于类型转换失败时给出提示信息，内容如下。

```
#改变默认的类型转换失败后的提示信息
xwork.default.invalid.fieldvalue={0}invalid field value for field!
```

继续新建 globalMessages_zh_CN.properties 文件，即中文环境的国际化资源文件，内容如下。

```
xwork.default.invalid.fieldvalue={0}字段类型转换失败！
```



这一步和 3.4.4 节中的第 4、5 步一样，这里可以直接使用 3.4.4 节案例中的国际化资源文件，内容不变。

(7) 在 struts.xml 文件中配置 RegistAction 类，并配置类型转换失败后跳转至“input”的逻辑视图。配置如下。

```
<!-- 通过常量配置 Struts2 所使用的解码集 -->
```

```
<constant name="struts.i18n.encoding" value="gbk" />
<constant name="struts.devMode" value="true" />
<constant name="struts.custom.i18n.resources" value="globalMessages"/>
<package name="convert" namespace="/convert" extends="struts-default">
    <action name="regist" class="com.struts2.actions.RegistAction">
            <result name="success">success.jsp</result>
            <result name="input">/regist.jsp</result>
    </action>
</package>
```

(8) 在 WebRoot 下新建 regist.jsp 页面，创建一个表单及三个表单元素，内容如下。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@taglib prefix="s" uri="/struts-tags" %>
<form action="convert/regist.action" method="post">
    <table align="center" width="360">
        <tr>
            <td>用户名</td>
            <td><input name="user.username" type="text">

        </tr>
        <tr>
            <td>年龄：</td>
            <td><input name="user.age" type="text">
        </tr>
        <tr>
            <td>出生日期：</td>
            <td><input name="user.birth" type="text">
        </tr>
        <tr align="center">
            <td><input type="submit" value="注册"/><input type="reset" value="
重设"/></td>
        </tr>
    </table>
<form>
```

(9) 创建注册成功页面 success.jsp，输出注册信息。使用<s:property .../>标签即可输出。

3.5.4　运行结果

运行 regist.jsp 页面，在页面的三个文本框中分别输入：lizanhong、24、19860221，如图 3-14 所示。

由于出生日期填写的并不是“yyyy-MM-dd”的日期格式，因此类型转换失败。单击“注册”按钮，跳转至 input 逻辑视图 regist.jsp 页面，即本页面，提示类型转换失败的信息，如图 3-15 所示。

图 3-14　注册界面　　　　图 3-15　类型转换失败

3.5.5　实例分析



源码解析：

上面的实例在 User 类中对 setAge()方法和 setBirth()方法进行了类型转换注解，当用户在表单中输入年龄和出生日期后，系统将在对 User 类中的 age 属性和 birth 属性进行赋值(执行 setXxx()方法)时，先执行类型转换器：IntegerTypeConverter 类和 DateTypeConverter 类，如果转换失败，加载国际化资源文件，提示类型转换失败信息，同时控制台输出异常信息。

3.6　常见问题解答

3.6.1　有关 Struts 2 中的 java.util.Date 类型转换的问题



有关 Struts 2 中的 java.util.Date 类型转换的问题！
网络课堂：http://bbs.itzcn.com/thread-10997-1-1.html

使用 Struts 2 开发 Web 应用时，如果使用 Date 类型数据时，Struts 2 会调用 Date 转换器来处理页面的 Date 字符串到 Date 类型的转换。阅读了 Struts 2 的 Code 后，发现 java.util.Date 转换器使用的日期格式都不是中文日期格式，而我们的日期字符串格式为 2010-11-15 17:14，当找不到对应的日期格式时，Struts 2 会用短日期格式来处理(yyyy-MM-dd)，这时如果需要长日期格式的数据，在日期格式转换后是获取不到准确数据的，遇到这种问题时，着急该怎么办呢？

【解决办法】：如果想要的值是 2010-11-15 17:14，而获取到的值是 2010-11-15，没有了时间，常有两种解决办法。第一种就是将日期类型改为字符串类型，在业务处理时转换；第二种是在日期的 getXxx()方法中增加上当前时间。

3.6.2　怎么自定义 struts 2 类型转换的全局与局部错误信息



怎么自定义 struts 2 类型转换的全局与局部错误信息?
网络课堂：http://bbs.itzcn.com/thread-10998-1-1.html

遇到类型转换错误的时候(也就是说不能进行类型转换)，Struts 2 框架会自动生成一条错误信息，并且将该错误信息放到 addFieldError()方法里面。此时，可以通过配置文件来替换这条由 Struts 2 自动生成的错误信息。

1．类型转换全局错误信息的设定

(1)　在 struts.xml 文件中添加以下代码。

```
<constant name="struts.custom.i18n.resources" value="message"></constant>
```

其中，name 的值是固定的，value 的值表示国际化资源文件的文件名为 message.properties。

(2)　生成 message.properties 文件，存放目录为 classes 的根目录。message.properties 文件内容如下。

```
xwork.default.invalid.fieldvalue={0} error
```

其中，“xwork.default.invalid.fieldvalue”为固定的，“{0}”相当于一个占位符，会被页面表单中元素的名字填充，是动态的。xwork.default.invalid.filedvalue 的值表示当前整个项目任何一个属性转换失败时，会提示“{0}(即属性的名字)error”。

2．类型转换局部错误信息的设定

在要进行类型转换的类的同级目录下新建 ClassName.properties 文件，设置类属性转换失败提示信息，代码如下。

```
invalid.fieldvalue.age=age is error
invalid.fieldvalue.birthday=birthday is error
invalid.fieldvalue.graduation=graduation is error
```

文件名中的 ClassName 要与进行类型转换的类名称一样，但扩展名为 properties，存放路径要与进行类型转换的类相同。其中的“age”、“birthday”和“graduation”与进行类型转换的类的属性名对应，后面为类型转换失败后的提示信息。

3.6.3　自定义 Struts 2 中类型转换失败提示信息问题



自定义 Struts 2 中类型转换失败提示信息问题!
网络课堂：http://bbs.itzcn.com/thread-10999-1-1.html

新建了 ClassName-properties 文件将框架自带的信息替换掉，可是发现不支持中文，而页面中设置的支持中文！此事该如何解决?

【解决办法】：如果你是熟手，这样的问题对你来说是最简单不过了，properties 文件不支持中文，需要把中文转换成相应的 Unicode 码，JDK 里面有转换工具。

3.6.4 Struts 2 标签<s:datetimepicker>中获取到的日期格式如何转换



Struts 2 标签<s:datetimepicker>中获取到的日期格式如何转换？
网络课堂：http://bbs.itzcn.com/thread-11000-1-1.html

我对 Struts 2 标签<s:datetimepicker>不是很了解，但是每次遇到时间类型的我都用这个标签来解决，得到的日期格式是：Sun May 04 00:00:00 CST 2010，需要转换为“2010-05-04 09:48:17.687”格式，怎么解决啊？

【解决办法】：解决这类问题有两种方法：第一种方法是在 Action 类中加入 DateFormat. getDateInstance(). format(new Date());获取当前时间即可；第二种方法是可以在页面中加入以下代码。

```
<s:head theme=”ajax”/>
```

这样，就可以在页面中使用标签了，如下面代码所示。

```
<s:datetimepicker name="todayDate" value="2010-05-04" label="Format
(yyyy-MM-dd)" displayFormat="yyyy-MM-dd"/>
```

3.7 习　　题

一、填空题

(1) Web 开发中，很多情况下需要使用自定义类型转换器，而使用自定义类型转换器时，需要自定义转换器类，并且需要把自定义的转换器类注册到 Web 程序中，注册方式有三种，分别是：注册局部类型转换器、注册全局类型转换器和__________。

(2) 比如一个程序中有一个 com.struts2.model.Student 类，它有三个属性，分别是 name、age 和 sex，在 com.struts2.actions.StudentAction 类中有如下属性：

```
private Set students;
```

其中，Set 集合中的元素类型为 com.struts2.model.Student 类。在局部类型转换注册文件 StudentAction-conversion.properties 中添加__________代码，可以指定 Set 集合元素的索引属性名为 Student 类中的 name 属性。

(3) Struts 2 提供类型转换异常处理机制，提供名称为__________的拦截器，这个拦截器被注册在默认拦截器栈中。

(4) Struts 2 允许改变类型转换失败后的提示信息，只要在应用的国际化资源文件中增加如下一行代码即可

__________={0}字段类型转换失败！

二、选择题

(1) 局部类型转换器的注册方式需提供文件名为__________格式的文件。

A. ActionName-conversion.properties

B. ClassName-conversion.properties

C. ActionName.properties

D. ClassName.properties

(2) 例如存在一个 com.struts2.model.Student 类，它有三个属性，分别是 name、age 和 sex，并有相应的 setter、getter 方法，若 com.struts2.actions.StudentAction 有如下属性。

```
private List students;
```

那么如何指定 List 集合元素为 com.struts2.model.Student 类型呢？__________

A. 在 com.struts2.actions 包下新建 StudentAction-conversion.properties 文件(即 StudentAction 的局部类型转换注册文件)，内容如下。

```
Element_students=com.struts2.model.Student
```

B. 在 com.struts2.actions 包下新建 StudentAction-conversion.properties 文件(即 StudentAction 的局部类型转换注册文件)，内容如下。

```
Element_Student =com.struts2.model.Student
```

C. 在 com.struts2.actions 包下新建 StudentAction-conversion.properties 文件(即 StudentAction 的局部类型转换注册文件)，内容如下。

```
students=com.struts2.model.Student
```

D. 在 com.struts2.actions 包下新建 StudentAction-conversion.properties 文件(即 StudentAction 的局部类型转换注册文件)，内容如下。

```
Student =com.struts2.model.Student
```

(3) 在使用 TypeConversion 注解时，设置 type 为：type=ConversionType.CLASS，转换规则将从一个名为__________文件中转配，该文件和其相关的 Action 类在同一个包中。

A. xwork-conversion.properties

B. ClassName.properties

C. ClassName-conversion.properties

D. ActionName-conversion.properties

三、上机练习

上机练习：计算长方形的面积。

要求：在页面中输入长方形的长和宽(如 20,15)，之间以逗号(,)隔开，如图 3-16 所示。如果输入的数据不是整数类型，则返回 input 逻辑视图，并提示类型转换失败信息，如图 3-17 所示。



图 3-16　输入长方形的长和宽



图 3-17　类型转换失败提示信息

如果输入的长和宽都是整数类型，单击“确定”按钮，表单提交至 RectangleAction 类的 execute()方法中，在执行此方法之前，先对用户输入的数据进行类型转换，转换成功计算长方形的面积，返回 success 逻辑视图，界面如图 3-18 所示。



图 3-18　计算长方形面积的结果界面



第 4 章　国际化与异常处理

内容摘要：

在 Java 语言国际化 API 中，影响数据本地化的因素主要有两个：一个是用户的语言环境，另一个就是用户的时区。语言环境表示某一特定区域或文化的语言习惯，包括时间数字和货币金额的格式。时区是数据本地化中的第二个因素，这是因为一些语言环境分布地理区域广泛。由于地球的自转，造成世界上的各个国家或地区在时间上有差异，而且各个国家或地区的日期显示方式也不一样。聪明的 JSP 开发人员早已解决了这个问题！对于这个问题的解决方法，将是本章所介绍的知识内容。

学习目标：

- Java 国际化的思路。
- Struts 2 中的全局国际化资源文件。
- 输出国际化消息。
- 使用 Action 范围的国际化。
- 使用 i18n 标签实现国际化。
- 使用 Struts 2 实现国际化。

4.1 国际化基础

互联网的出现将不同国家、不同地区的人们联系到了一起，也为使用互联网的人们带来了各种信息。由于地域不同，不同民族使用的语言也存在很大的差异。因此就出现一个问题，如何才能使不同区域的用户都可以读懂同一个 Web 页面为大家提供的信息呢？例如，一个中国用户如何能够轻松阅读一个由英国用户提供的页面呢？通过使用国际化这一手段，可以实现这一目的。

信息的国际化，可以动态构建一个具有各种不同语言版本的 Web 应用程序，成为面向国际应用的 Web 页面。当然，这一功能需要利用 Java 语言的 Unicode 字符集。本节将向读者介绍利用 Struts 2 实现国际化的基础知识。

视频教学： 光盘/videos/04/Internation1.avi　　长度：7 分钟
光盘/videos/04/Internation2.avi　　长度：6 分钟
光盘/videos/04/Internation3.avi　　长度：6 分钟
光盘/videos/04/Internation4.avi　　长度：9 分钟

4.1.1 基础知识——国际化与本地化

国际化(Internationalization)是设计和制造容易适应不同区域要求的产品的一种方式。它要求从产品中抽离所有的与语言、国家/地区和文化相关的元素。换言之，应用程序的功能和代码设计考虑在不同地区运行的需要，其代码简化了不同本地版本的生产。开发这样程序的过程，就称为国际化。

本地化(Localization)是为解决网站、软件向其他国家推广时遇到的语言障碍问题。网站需要翻译成不同国家的语言，以便不同国家的人能够无障碍地阅读网站内容，这便是网站本地化；软件也需要本地化，以便能够在目标国家推广。本地化不仅仅是简单的文字翻译转换，还必须根据目标语言国家的市场特点、文化习惯、法律等情况进行本地特性开发、界面布局调整等工作。

下面将讲解一下国际化过程中使用到的一些基础知识。

4.1.2 基础知识——Locale 类

Locale 类有 Locale(String langeuage)和 Locale(String language,String country)两个常用的构造方法。其中 langeuage 表示语言，它的取值范围在 ISO-639 定义的小写的、两个字符组成的语言代码。country 表示国家或者地区，它的取值范围是 ISO-3166 定义的大写的、两个字符组成的代码。表 4-1 和表 4-2 列出了 ISO-639 语言代码和 ISO-3166 国家区域代码。

表 4-1　常用的 ISO-639 语言代码

语　言	代　码
汉语	zh
英语	en
日语	ja
德语	de

表 4-2　常用的 ISO-3166 国家和地区代码

国　家	代　码
中国	CN
美国	US
英国	BG
日本	JP
德国	DE

Locale 的应用语法如以下代码所示。

```
Locale locale = new Locale("zh","CN");
```

为了方便开发人员的理解开发，在 Locale 类中还定义了许多 Locale 对象的常量，供开发人员开发。主要应用于国家和地区的 Locale 对象如以下代码所示。

```
Locale.CHINA          //中国
Locale.US             //美国
Locale.UK             //英国
Locale.CANADA         //加拿大
Locale.FRANCE         //法国
Locale.ITALY          //澳大利亚
Locale.KOREA          //韩国
Locale.PRC            //中华人民共和国
Locale.JAPAN          //日本
```

同样，Locale 对象还提供了常用的语言常量，详细如以下代码所示。

```
Locale.CHINESE
Locale.ENGLISH
Locale.JAPANESE
Locale.FRENCH
Locale.TRADITIONAL CHINESE
Locale.SIMPLIFIED_CHINESE
```

提示

在 Locale 类中，还定义了一个静态的方法 getDfault()，用于获取本地系统默认的 Locale 对象。要查看 Java 支持的所有语言环境，可以调用 Locale 类中的静态方法 getAvailableLocales()函数，该函数返回一个 Locale 对象数组。

4.1.3 基础知识——资源包

在编写应用程序的时候，需要面对的一个问题是如何处理与 locale 相关的一些信息。例如，页面上的一些静态文本能够以用户习惯的语言显示。最原始的做法是将这些信息编码到程序中(可能是一大串判断语句)，但是这样就将程序代码和易变的 locale 信息捆绑在一起，以后如果需要修改 locale 信息或者添加其他的 locale 信息，就显得不太方便。而资源包可以帮助解决这个问题，它通过将可变的 locale 信息放入资源包来达到两者分离的目的。应用程序可以自动地通过当前的 locale 设置到相应的资源包中取得所要的信息。资源包的概念类似于 Windows 编程人员使用的资源文件(rc 文件)。

获取资源包有以下几种不同的获取方法。我们需要调用 ResourceBundle 类中的静态方法 getBundle()。

- 根据基名和得到以资源包，使用系统缺省的 Locale 对象，语法如下所示。

```
public static final ResourceBundle getBundle(String baseName)
```

- 根据基名和 Locale 对象获取资源包，语法如下所示。
  该方法返回的是一个 Locale 对象，在上面已经提到需要使用 getBundle()方法，只是参数不同而已，故关系明确。

```
public static final ResourceBundle getBundle(String baseName,Locale
locale)
```

- 从资源包中根据关键字得到值。
  利用 ResourceBundle 类的 getObject()方法，还可以从资源包中得到任何需要的对象，语法如下所示。

```
public final Object getObject(String key)
```

利用 getBundle()方法可以得到对应于某个 Locale 对象的资源包，然后利用 ResourceBundle 类的 getString()方法得到相应语言版本的字符串，语法如下所示。

```
public final String getString(String key)
```

如果在使用资源类的时候需要扩展资源类，那么必须要扩展 ResourceBundle 类，这个时候需要实现下面两个方法。

◆ 返回资源包的关键字枚举，语法如下所示。

```
    public abstract Enumeration getKeys()
```

◆ 从资源包中根据关键字得到对象，语法如下所示。

```
    protected abstract Object handleGetObject(String key)
```

为了方便开发人员的编写和使用，在 java.util 包中还提供了 ListResource Bundle 和 PropertyResourceBundle 两个资源类，它们都是从 ResourceBundle 类中派生出来的。

在编程的时候如果想使用 ListResourceBundle 类，只需要将所有的资源放入一个对象数组即可，它还提供了查找资源的功能。

如果全部的资源都是字符串类型，可以使用 PropertyResourceBundle 类。该类对于不同的国家分别提供不同的文件，它们的属性文件名称都有一定的规律，都是资源类的命名方式，它们的扩展名都是.properties，文件的内容都是以键值对的形式存储的。

在加载资源的时候可以使用 ResourceBundle 类的 getBundle()静态方法。该方法执行顺序是，先加载资源类，如果没有成功则加载属性资源文件；如果成功则创建 Property ResourceBundle 对象。

getBundle()静态方法将会按照下列顺序查找资源包。

- ChinaResource_zh_CN.class
- ChinaResource_zh_CN.properties
- ChianaResource_zh.class
- ChinaResource_zh.properties
- ChinaResource.class
- ChinaResource.properties

不同的语言编写可以对应不同的资源文件，对应不同的程序，非常有利于开发者开发项目。

在属性文件中保存了 7 位 ASCII 码字符，对于中文字符需要通过转换为相应的 Unicode 码，其格式为\uXXXX。在 JDK 的开发工具包中，可以通过 native2ascii 工具将非 ASCII 字符转换为 Unicode 编码。

4.1.4　基础知识——加载资源文件的顺序

在 Struts 2 框架中，可以设置多种形式的资源文件，这些资源文件具有一定的优先级，这个优先级，也就是 Struts 2 框架加载资源文件的顺序。对国际化信息的加载情况，可以分为两类，在 Action 中和在 JSP 文件中。

1．在 Action 中

如果国际化信息在 Action 类中，这时加载资源文件的顺序如下。

(1)　优先加载 Action 范围资源文件，即在 Action 所在目录下，basename 为该 Action 类名的资源文件。

(2)　如果在(1)中找不到指定 Key 值，则加载其父类 Action 的资源文件。例如存在父类 Action，其类名为 FatherName，则查找 basename 为 FatherName 的资源文件。

(3)　如果在(2)中找不到指定 Key 值，则加载所实现的接口类的资源文件。例如 Action 实现接口 InterfaceName，则查找 basename 为 InterfaceName 的资源文件。

(4)　如果在(3)中找不到指定 Key 值，且 Action 实现接口 ModelDriven，则使用 getModel()方法返回的 model 对象，重新执行(1)。

(5)　如果在(4)中找不到指定 Key 值，则查找当前包下，basename 为 package 的资源文件。

(6)　如果在(5)中找不到指定 Key 值，则沿着当前包向上查找，查找 basename 为 package 的资源文件，直到最顶层包。

(7)　如果在(6)中找不到指定 Key 值，则获得 struts.custom.i18n.resources 常量所指定的 value 值，加载 basename 为该 value 值的资源文件。

(8) 如果在上面的步骤中仍然找不到指定 Key 值，则直接输出这个 Key 的字符串值。



提示

如果在从(1)到(7)的步骤中，找到指定 Key 对应的值，则系统停止查找，然后输出 Key 对应的资源信息。

2. 在 JSP 文件中

如果国际化信息在 JSP 文件中通过 text 标签或者表单标签来定义，这时加载资源文件的顺序，分为以下两种情况。

1) 使用 Struts 2 的 i18n 标签

(1) 从 i18n 标签指定的国际化资源文件中，加载指定 Key 对应的值。

(2) 如果在(1)中找不到指定 Key 值，则查找 struts.custom.i18n.resources 常量所指定的 value 值，加载 basename 为该 value 值的资源文件。

(3) 如果在上面的步骤中仍然找不到指定 Key 值，则直接输出这个 Key 的字符串值；如果在前面的任意一个步骤中，找到指定 Key 对应的值，则系统停止查找，然后输出 Key 对应的资源信息。

2) 没有使用 Struts 2 的 i18n 标签

(1) 这时的 text 标签和表单标签，如果没有被包含在 i18n 标签中，则直接查找 struts.custom.i18n.resources 常量所指定的 value 值，加载 basename 为该 value 值的资源文件；如果找到对应 Key 值，则输出 Key 对应的资源信息。

(2) 如果在(1)中找不到指定 Key 值，则直接输出这个 Key 的字符串值。

4.2 将用户注册国际化

本节主要讲解将用户注册程序改变为一个国际化的用户注册程序。不同的用户浏览器语言环境不同，访问显示的页面语言也不同，接下来就如何实现国际化注册程序的步骤给以讲解。

视频教学：光盘/videos/04/config1.avi　　长度：6 分钟
　　　　　光盘/videos/04/config3.avi　　长度：11 分钟

4.2.1 基础知识——国际化的配置文件

Struts 2 框架将国际化信息定义在资源文件中，在配置文件中对资源文件进行配置，告诉 Struts 2 框架所需要加载的资源文件。在不同的配置文件中，对资源文件的配置方式不同。

1. 使用 struts.xml 文件

在 struts.xml 文件中配置国际化资源文件，一般是通过设置常量来实现。例如，配置一个 basename 为 globalMessages 的国际化资源文件，代码如下。

```
<constant name="struts.custom.i18n.resources" value="globalMessages"/>
```

2．使用 struts.properties 文件

如果在 struts.properties 文件中实现，则需要使用 Key-Value 的代码格式，配置代码如下。

```
struts.custom.i18n.resources=globalMessages
```

3．使用 web.xml 文件

如果在 web.xml 文件中实现，则需要使用<param-name>和<param-value>元素，配置代码如下。

```
<init-param>
    <param-name>struts.custom.i18n.resources</param-name>
    <param-value>globalMessages</param-value>
</init-param>
```



配置 Struts 2 国际化资源文件时，最好是在 struts.xml 或者 struts.properties 文件中实现。

通过上述三种形式的配置代码，可以发现，配置 Struts 2 国际化资源信息需要两个值：struts.custom.i18n.resources 和 globalMessages。

(1) struts.custom.i18n.resources：在 Struts 2 框架中，表示国际化的常量，是一个固定不变的值。

(2) globalMessages：表示全局国际化资源文件的 basename 值，对应的全局国际化资源文件，可以是 globalMessages_ zh_CN.properties、globalMessages_en_US.properties 等，这些文件名都符合 globalMessages_ language_country 的格式。

4.2.2　基础知识——在文本中使用参数

Struts 2 为开发者提供了两种不同的在消息中设置参数的方式。

1．占位符的方式

占位符的方式就是 Java 中设置消息文本参数的方式，相当于使用从{0}到{9}的占位符。当使用 MessageFormat 类的 format()方法格式化消息字符串时，参数信息内容传递进来，替换掉消息文本中的占位符。

2．OGNL 表达式

使用 OGNL 表达式方式，不同于在标签的属性中使用的 OGNL 表达式，在消息文本中使用 OGNL 表达式的语法是：${ms}。

例如，设置用户登录时提示用户登录后的欢迎消息，资源文件代码如下所示。

```
ms=${username},您好！欢迎登录窗内网个人空间！
```

在登录成功首页面内设置 text 标签，来输出资源文件中的消息内容，代码如下所示。

```
<s:text name="ms">
```

获取键值为 ms 的消息文本时，${}中的表达式 ms 将会根据栈自动进行计算，最后显示到页面中替换消息后的内容信息。

在消息文本中使用数字占位符可以看成是被动地接受值，而 OGNL 表达式则可以看成是主动地获取值。

4.2.3 基础知识——访问国际化消息

在 Java 国际化中，使用 MessageFormat 类来处理占位符。Struts 2 作为一个优秀的 MVC 框架，为占位符提供更加简单的操作方式。

在 Struts 2 框架中，访问国际化消息主要有如下两种情况。

- 在 JSP 文件中：使用<s:text name="key"/>，或者为表单元素指定一个 key 属性。
- 在 Action 类中：使用 ActionSupport 类的 getText()方法。

所以，要处理 Struts 2 中的占位符，也有以下两种情况。

- 在 JSP 文件中：在 Struts 2 框架的 text 标签中，使用 param 标签引用国际化资源中的占位符。一个 param 标签对应一个占位符。
- 在 Action 类中：使用 ActionSupport 类中的 getText()方法。根据占位符的个数和类型，getText()方法的声明有多种方式，如表 4-3 所示。

表 4-3 getText()方法的声明方式

声明方式	说 明
String getText(String keyName)	以国际化资源文件中的 Key 作为参数
String getText(String keyName,List args)	以 List 集合作为第二个参数，集合中的每个元素分别对应 keyName 中的一个占位符
String getText(String keyName,String[] args)	以字符串数组作为第二个参数，数组中的每个元素分别对应 keyName 中的一个占位符
String getText(String keyName,String defaultValue, List args)	如果找不到国际化资源信息，则返回 defaultValue 对应的内容
String getText(String keyName,String defaultValue, String[] args)	如果找不到国际化资源信息，则返回 defaultValue 对应的内容

1. 在 JSP 页面中访问本地化消息

经常使用 Struts 2 的读者都知道，它为我们提供了一个<s:text/>标签，用于访问本地化消息。在 JSP 页面中可以通过 text 标签访问键为 message 的消息内容。例如：

```
message=登录成功
```

```
<s:text name="message">
```

在使用过程中，如果传输的消息内容有参数，可以通过使用嵌套的 param 标签设置需要填充的参数信息。例如：

```
<s:text name="message">
<s:param value="段留治"/>
</s:text>
message={0}，欢迎您登录我们的系统。
```

在上述代码中，看到的 param 标签的顺序和在消息文本中的占位符顺序是相同的，其中 param 标签替换文本文件中{0}占位符。但是在 text 标签中最多只能嵌套 10 个 param 参数标签。

2．在 Action 中访问本地化消息

在 Struts 2 中使用在 Action 中访问本地化消息，首先要知道 Struts 2 中 TextProvider 接口中定义了要访问本地化消息的方法，同时 ActionSupport 类也可以实现该接口。所以在编写 Action 类继承自 ActionSupport 类时，可以直接使用这些方法。

此外，ActionSupport 提供了很多重载的 getText()方法，用来访问本地化信息，接下来列举几个常用的 getText()方法。

```
public String getText(String textName)
```

参数 textName 是键的消息内容，如果没有找到，则返回 null。

```
public String getText(String textName,String defaultName)
```

参数 textName 和上述一样，如果没有查找到，则返回 defaultName 信息。

```
public String getText(String textName,List args)
```

参数 textName 和上述一样，而 args 表示用来代替列表消息中的占位符，第一个参数替换占位符{0}，依次类推。

```
public String getText(String textName,String[] args)
```

参数 textName 和上述一样，参数 args 和上述功能几乎相似，不同的是该参数是一个 String 类型的数组，表示用来替换数组中的消息占位符，第一个参数替换占位符{0}，依次类推。接下来用一个简单的例子解释如何使用 getText()重载方法。

在资源配置文件中写入如下内容：

```
message={0}您好，欢迎您来到我们窗内网。您的等级为{1}
```

在 Action 中可以使用上面讲到的 getText()重载方法中的最后一种数组重载，代码如下所示。

```
String msg=getText("message",new String[]{"段留治",20});
```

在代码中声明 msg 接受返回的消息字符串，同时使用了 getText()的重载方法，第一个参数是键的消息内容，第二个参数是一个 String 类型的数组参数，表示占位符。

3．在表单标签属性中访问本地化消息

使用表单标签时，需要注意的是，label 属性的值通常都是从资源文件中获取的。在 label 属性中，可以通过调用 getText()方法来获取消息的内容，代码如下所示。

```
<s:textfield name="user.name" label="%{getText('username')}">
```

此外，还可以使用 Struts 2 表单标签中的 key 属性来设置消息的内容 key，key 可以使用 label 属性自动生成的值，代码如下所示。

```
<s:textField name="user.name"key="name">
```

4．在资源文件中访问本地化消息

在资源文件中，可以使用 OGNL 表达式，还可以使用 OGNL 表达式在一个消息文本中访问本地化消息。使用 OGNL 表达式${getText("key")}访问本地化消息，可以在两个消息文本之间相互访问，这为我们提供了非常大的方便。

```
email=某某邮箱
error.email.invalid=${getText("email")}邮箱格式错误
```

4.2.4 实例描述

本实例我们通过使用资源文件将注册页面国际化，通过使用 Properties Editor 工具将资源文件中的中文编码转换为 Unicode 编码格式以供程序使用。当浏览器访问该页面时，通过浏览器的语言环境来调用不同的资源文件，从而完成想要的注册页面国际化功能。下面是创建国际化的主要步骤。

4.2.5 实例应用

【例 4-1】 用户注册页面国际化。

查看用户注册页面，将所需要显示的文字内容(例如“用户名”)分离，保存到资源文件内。创建一个与 RegisterAction 同名的资源文件，这个资源文件只能通过 RegisterAction 来访问，该文件只需要保存显示注册页面的文字内容。创建一个 RegisterAction_ zh_CN.properties 文件，代码如下所示。

```
title=\u7528\u6237\u6CE8\u518C
username=\u7528\u6237\u540D
password=\u5BC6\u7801
sex=\u6027\u522B
sex.male=\u7537\u6027
sex.female=\u5973\u6027
email=\u90AE\u7BB1
pwdQuestion=\u5BC6\u7801\u95EE\u9898
pwdAnswer=\u5BC6\u7801\u7B54\u6848
submit=\u6CE8\u518C
reset=\u91CD\u7F6E
success=\u6CE8\u518C\u6210\u529F\uFF01
success.info=\u795D\u8D3A\uFF0C${user.username}\uFF0C\u6CE8\u518C\u6210\
u529F\uFF01
failure=\u6CE8\u518C\u5931\u8D25
failure.info=\u6CE8\u518C\u5931\u8D25,\u56E0\u4E3A\uFF1A${exception}.<br/>\
u8BF7<a href\="{0}">retry</a>
```

在上述代码中，通过使用 Properties Editor 将需要显示的文字转换为 Unicode 编码格式，并保存在该资源文件中。在 success.info 和 failure.info 的消息文本中使用了 OGNL 表达式，success.info 获取 User 对象的 username 属性，failure.info 则是发生异常时输出栈中的异常对象。

访问 RegisterAction.java 的 Action 类，代码如下所示。

```
public class RegisterAction extends ActionSupport{
    public String execute()
    {
        return SUCCESS;
    }
}
```

上述代码中，编写了 execute()默认方法，当客户端发送请求到该页面时调用该方法，返回到注册页面。

下面是使用 Struts 2 提供的访问国际化消息的方式，显示页面信息内容，编写注册页面，通过 Struts 2 中的标签进行显示，详细代码如下所示。

```
<s:form>
    <s:textfield name="user.username" key="username"></s:textfield>
    <s:password name="user.password" key="password"></s:password>
    <s:radio name="user.sex" value="true"
list="#{true:getText('sex.male'),false:getText('sex.female')}"
key="sex"></s:radio>
    <s:textfield name="user.email" key="email"></s:textfield>
    <s:textfield name="user.pwdQuestion" key="pwdQuestion"></s:textfield>
    <s:textfield name="user.pwdAnswer" key="pwdAnswer"></s:textfield>
</s:form>
```

上述代码中，通过 Struts 2 提供的丰富的组件标签，(后面将进行详细的讲解)设置标签的 key 属性，显示资源文件的文字信息。

4.2.6　运行结果

启动 Tomcate 服务器，输入地址 http://localhost:8080/ch4_2/regist.action，访问国际化后的注册页面，运行结果如图 4-1 所示。

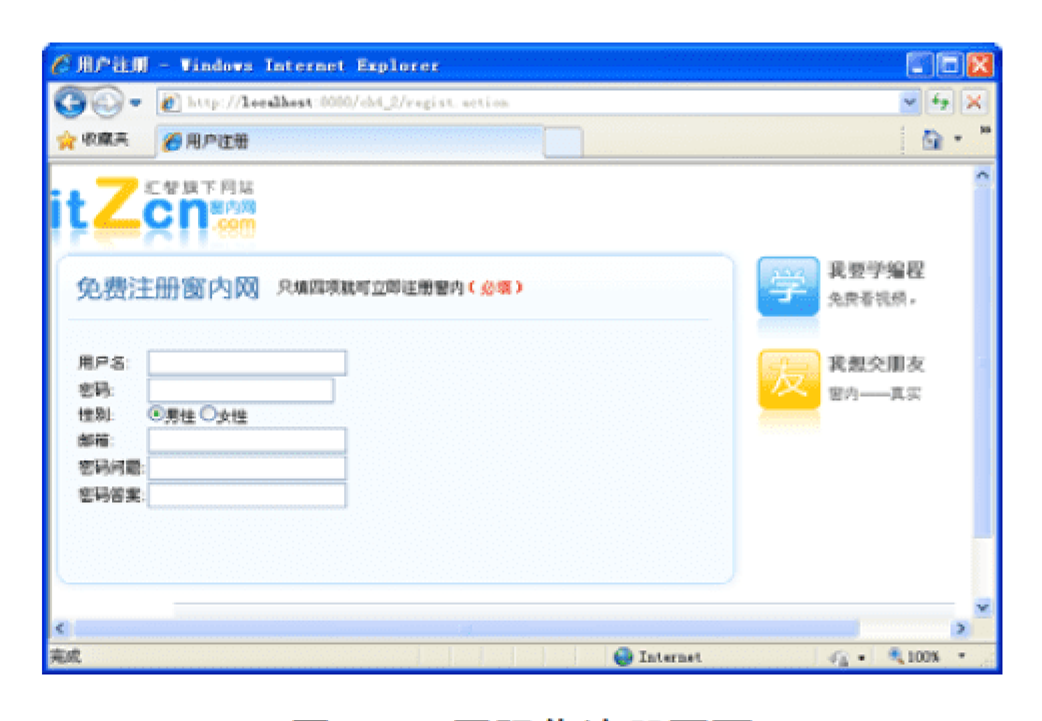


图 4-1　国际化注册页面

4.2.7 实例分析



源码解析：

通过上面的实例，可以学到如何在消息文本中使用 OGNL 表达式。通过获取相应的属性信息，在页面上使用 Struts 2 中的标签，获取资源文件中的属性值和内容，完成注册页面国际化。

4.3 消息提示国际化

制作一个 Web 程序时，如果发生了错误系统都会为用户提供错误信息。如果错误出现在一个英文环境下的浏览器中，对于不懂得英语的用户来说可能是件头疼的事。但如果将错误信息编写在程序中，则是个十分合理的解决办法。这样可以方便用户使用和阅读错误信息，从而找到出错原因。本节的消息提示国际化可以帮助用户来实现这一功能。



视频教学：光盘/videos/04/config2.avi　　长度：6 分钟

4.3.1 实例描述

用户注册时判断该用户名是否存在，如果该用户重复登录会使系统抛出异常，则使用 Action 获取捕获的异常，然后从定义的消息资源文件中获取提示的错误信息，并使用标签将消息显示在页面上。

4.3.2 实例应用

【例 4-2】 用户注册时判断该用户名是否存在。

首先在 src 工程目录下创建 RegisterActin_zh_CN.properties 文件，该文件用来显示用户名存在的信息提示，详细代码如下所示。

```
error.username.exist=\u7528\u6237\u540D\u4EE5\u5B58\u5728\uFF01
```

接下来定义一个异常处理类 UsernameExistException.java，该类继承自 Exception 类，详细代码如下所示。

```
public class UsernameExistException extends Exception{
}
```

该异常处理类的主要作用就是用来表示用户已经存在这种异常。

下面创建 RegisterAction 类，用来处理业务逻辑的 Action 类，该类中包含了一个 execute()方法，用来判断用户是否存在业务处理。当捕获到创建的 UsernameExistException 异常时，系统调用 addFieldError()方法，详细代码如下所示。

```
public class RegisterAction extends ActionSupport{

    private String username;
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    @Override
    public String execute() throws Exception {
        try{
            if(username.equals("admin"))
            {
                throw new UsernameExistException();
            }
        }catch(UsernameExistException e)
        {
            addFieldError("username", getText("error.username.exist"));
            return INPUT;
        }
        return SUCCESS;
    }
}
```

在上述代码中，获取用户表单提交的用户信息，使用 try 来捕获异常。如果用户名为 admin，则抛出异常，代码运行到 catch 时调用 addFieldError()方法。

最后设置修改显示提示信息页面，当用户名存在的时候，转交到信息提示页面，在该页面编写如下代码。

```
<body>
        <s:fielderror/>
</body>
```

使用 Struts 2 提供的 fielderror 异常标签来显示提示信息内容。

4.3.3　运行结果

完成上述代码后启动 Tomcat，运行 http://localhost:8080/ch4_3/，获得的结果如图 4-2 和图 4-3 所示。

图 4-2　注册页面

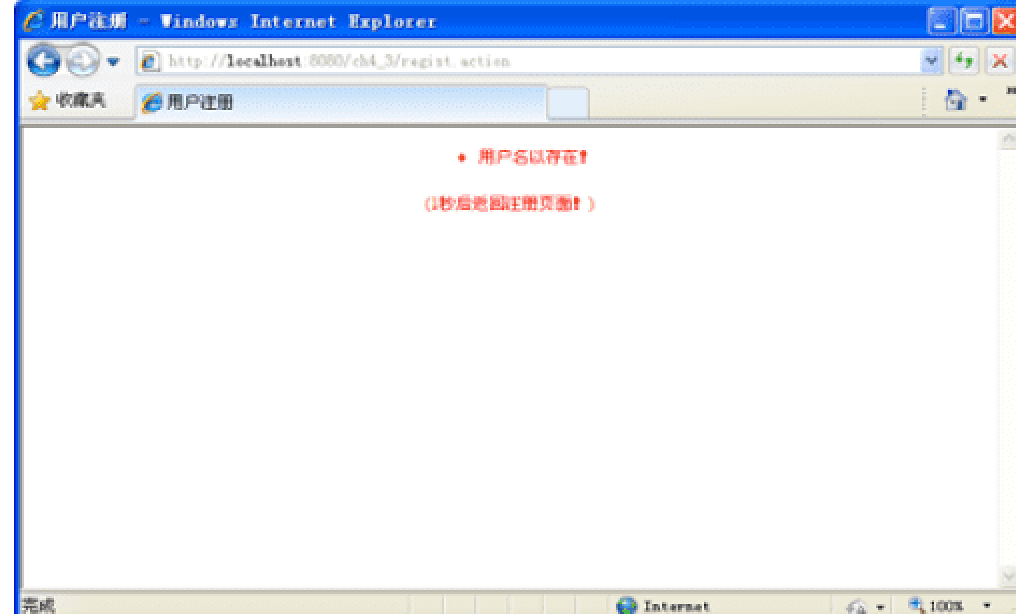


图 4-3　信息提示页面

4.3.4　实例分析



源码解析：

上述例子中，使用 Struts 2 中的<s:fielderror/>和异常处理机制捕获错误消息，提示用户“该用户名已经存在”。当获取到异常之后，使用 addFieldError("username", getText("error.username.exist"));的方法将信息获取返回到前台，并通过 Struts 2 显示到页面上。

4.4　手动改变注册页面国际化

没有学习使用国际化之前，如果完成一个中文和英文版的 Web 系统，需要开发两个同样的系统，然后将中文版的系统汉字全部翻译成英语，从而增加了工作的难度。如果学习了国际化相对来说就简单多了，只需要将资源文件中的中文字符串翻译成英语，保存到对应的英文 locale 的资源文件中即可。本节将向读者介绍一个手动改变注册页面国际化的典型应用，通过本案例的学习可以使读者掌握 locale 的使用方法。



视频教学： 光盘/videos/04/locale.avi　　　　**长度：** 8 分钟

4.4.1　基础知识——用户 locale 流程

国际化 Web 程序一般都有多个语言版本，用户可以通过语言版本选择首页。当用户访问时，首先给出这个首页，当用户选择某种语言之后，可以使用相应的语言类型更新首页。此外，还可以由程序根据浏览器发送的请求报头中的语言信息，来自动选择一种语言版本。

通常情况下会使用第二种方法来实现。下面讲解一下 Struts 2 如何设定访问用户的语言环境 locale。在加载资源文件时，Struts 2 会根据 ActionContext 的 getLocale()方法的返回值加载对应的资源文件。如果要改变当前的 locale，只需要重新调用 getLocale()方法并保存到 ActionContext 中即可。下面是设定 Struts 2 访问 locale 时的步骤。

(1) 判断 locale 属性是否有值，然后将值转换为 Locale 对象保存到 ActionContext 内。

(2) 当 locale 属性值为空或者没有设置的时候，则获取浏览器请求的报头信息内容，然后创建 Locale 对象并保存到 ActionContext 内。

(3) 通过拦截器获取 request_locale 请求参数的值，以该值创建一个 Locale 对象，将这个对象作为 session 的 WW_TRANS_I18N_LOCALE 属性，保存到 session 对象中，再把这个对象保存到 ActionContext 内。

4.4.2　实例描述

为了让用户更方便地使用中英文的切换，一般不用浏览器语言环境来变化，而是直接使用两个超链接“中文”和“English”，以方便用户随时查看不同版本的 Web 注册程序。当用户单击“中文”链接的时候，将在用户的 Web 页面上显示中文的信息内容；当用户单击 English 链接的时候，将在用户页面上显示英文版的 Web 注册系统。

实现该功能思路：当用户点击某个链接时发送 request_locale 请求参数。参数的内容是对应语言的 Locale 对象的字符串值。

4.4.3　实例应用

【例 4-3】 实现用户注册页面中英文版。

(1) 创建一个 Web 工程，在 com.itzcn.action 包下创建 RegisterAction_en.properties 和 RegisterAction_zh_CN.properties 语言国际化资源文件，用来定义用户界面显示中文版和英文版的信息内容的文件。

- RegisterAction_en.properties 资源文件内容如下。

```
#标题
title=User Register
#用户名
username=UserName
#用户密码
password=PassWord
#性别
sex=Sex
#男性
sex.male=Male
#女性
sex.female=Female
#邮箱
email=Email
#密码问题
pwdQuestion=Password Question
#密码答案
pwdAnswer=Password Answer
#提交
```

```
submit=Register
#重置
reset=Reset

success=Register success
success.info=${user.username},congratulation,register success!

failure=Register failure
failure.info=register failure,reason:${exception}.<br/>Please<a
href="{0}">retry</a>
error.username.exist=Username is already exist

#超链接文本
chinese=Chinese
english=English

#提示信息
selectlanguage=please select language
```

- RegisterAction_zh_CN.properties 资源文件内容如下。(转换为 Unicode 码后的内容):

```
title=\u7528\u6237\u6CE8\u518C
username=\u7528\u6237\u540D
password=\u5BC6\u7801
sex=\u6027\u522B
sex.male=\u7537\u6027
sex.female=\u5973\u6027
email=\u90AE\u7BB1
pwdQuestion=\u5BC6\u7801\u95EE\u9898
pwdAnswer=\u5BC6\u7801\u7B54\u6848
submit=\u6CE8\u518C
reset=\u91CD\u7F6E

success=\u6CE8\u518C\u6210\u529F\uFF01
success.info=\u795D\u8D3A\uFF0C${user.username}\uFF0C\u6CE8\u518C\u6210
\u529F\!\uFF01

failure=\u6CE8\u518C\u5931\u8D25
failure.info=\u6CE8\u518C\u5931\u8D25,\u56E0\u4E3A\uFF1A${exception}.
<br/>\u8BF7<a href\="{0}">retry</a>

error.username.exist=\u7528\u6237\u5DF2\u7ECF\u5B58\u5728\uFF01


chinese=\u4E2D\u6587
english=\u82F1\u6587

selectlanguage=\u8BF7\u9009\u62E9\u8BED\u8A00
```

(2) 创建处理提交的 Action 类，当用户单击中英文切换超链接时，转向指定的 RegisterAction.java 类中，在这里读者一定要注意的是：资源文件和 Action 类的名称前段部分要相同，详细代码如下所示。

```
public class RegisterAction extends ActionSupport{
    public String execute()
    {
        return SUCCESS;
    }
}
```

在该类中，RegisterAction 继承于 ActionSupport，创建 execute()的方法，当用户请求该类执行完毕时，转向用户的注册页面。

(3) 接下来编辑注册页面的信息内容，在该页面内主要用来显示用户界面的标题信息内容，该页面的内容信息从资源文件内读取，通过标签绑定到页面内，详细代码如下所示。

```
<!--顶部-->
<div id="header">
    <div class="h lt"><a href="http://www.itzcn.com"><img
src="images/logo.gif"/></a></div>
    <div class="h rt">
    <s:set name="current locale"
value="#session['WW TRANS I18N LOCALE']==null?locale:#session['WW TRANS
I18N_LOCALE']"/>
```

上述代码主要获取 locale。在页面第一次访问时，session 对象中的 WW_TRANS_I18N_LOCALE 属性为 null，因此表达式 locale 也就调用了 Action 类的 getLocale()方法，然后获取页面中的 locale 对象并保存到 current_locale 变量内。当用户选择语言超链接时，session 对象中就会存在一个 WW_TRANS_I18N_LOCALE 属性。下述代码则是设置中英切换的超链接内容。

```
<s:url id="chinese url" value="regist.action">
    <s:param name="request locale"
value="@java.util.Locale@CHINA"></s:param>
</s:url>
<s:url id="english url" value="regist.action">
    <s:param name="request_locale"
value="@java.util.Locale@ENGLISH"></s:param>
</s:url>
```

上述代码主要用来设置中英文切换的超链接。在这里使用到了 Struts 2 中的 URL 标签。该标签内部有个 param 标签，它用来对请求路径附加请求参数信息。OGNL 表达式是允许访问类的静态成员的。因此在设置 request_locale 请求参数时，可以直接使用 Locale 类的静态常量。



这里不能使用 Locale 类的 CHNESE 常量，该常量是表示的中文语言的 Locale 对象，而在简体中文系统中，Struts 2 初始构造的是表示中国的 Locale 对象。为了能在后面进行 Locale 对象的比较，我们使用 Locale.CHINA 常量。

以下代码则是用来输出不同环境下的中英文超链接代码。

```
<s:if test="#current locale.equals(@java.util.Locale@CHINA)">
    <s:a href="%{#chinese url}"><strong>
            <s:text name="chinese"></s:text>

    </strong></s:a><s:a href="%{#english url}"><s:text
name="english"></s:text> </s:a>
</s:if>
<s:else>
    <s:a href="%{#chinese url}"><s:text name="chinese"></s:text> </s:a>
    <s:a href="%{#english url}"><strong>
    <s:text name="english"></s:text>
    </strong></s:a>
</s:else>
</div>
<!--顶部-->
```

上述代码的主要作用是：输出中文和英文的超链接，使用 Struts 2 中的 if 和 else 标签进行判断，从而为当前的 Locale 对象对应的超链接添加样式。以下代码则是用来绑定注册页面提示信息、内容的，详细代码如下所示。

```
<div class="content">
    <p>
        <label for="email"><s:text name="username"></s:text>: </label>
        <INPUT class="input" type="text" id=username name="username">
        <span id="checkusername"> </span> </p>
    <p>
        <label for="email"> <s:text name="password"></s:text>: </label>
        <INPUT  class="input"  type=password  id="password"  name="userpass">
        <span id="checkpassword"> </span> </p>
    <p>
        <label for="email"> <s:text name="email"></s:text>: </label>
        <INPUT class="input" maxLength=32 name="userpass2">
        <span id="checkpassword2"> </span> </p>
    <p>
        <label for="email"><s:text name="pwdQuestion"></s:text>: </label>
        <INPUT id=email maxLength=32  class="input"  name=email>
        <span id="checkemail"> </span> </p>
    <p>
        <label for="email"> <s:text name="pwdAnswer"></s:text>: </label>
        <INPUT id=email maxLength=32  class="input"  name=email>
        <span id="checkemail"> </span> </p>
    <p>
        <INPUT type="submit" name="mysubmit" value="<s:text name="submit"/>">
        <INPUT type="submit" name="mysubmit" value="<s:text name="reset"/>">
    </p>
</div>
```

上述代码用来设置注册页面的注册提示信息的绑定，在这里同样使用 Struts 2 中的<s:text/>标签，将信息内容从不同的语言资源文件中读取出来，显示在该标签内。在前几章节中已经讲解到注册页面文本内容的国际化，这里就不再过多讲解。

4.4.4　运行结果

运行 Tomcate 服务器，启动项目，在浏览器地址栏中输入：http://localhost:8080/ch4_4/regist.action，显示注册页面运行结果如图 4-4 和图 4-5 所示。

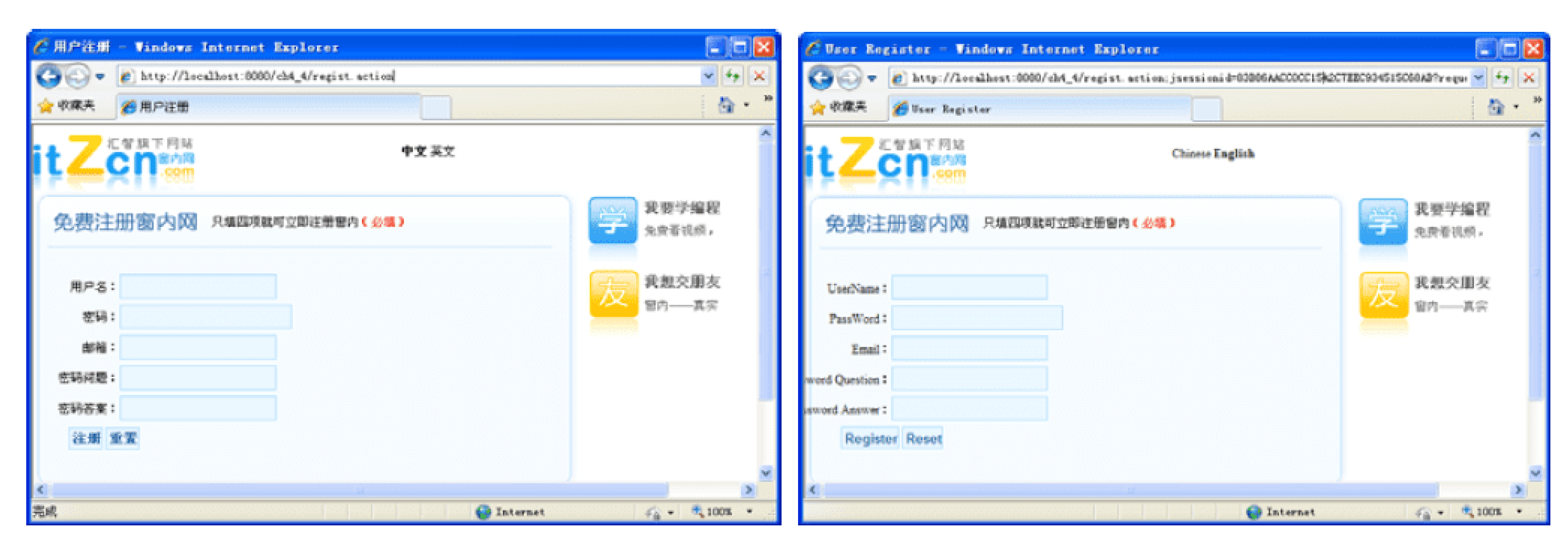
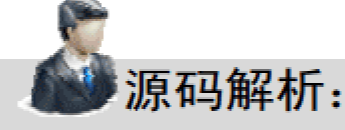


图 4-4　中文下的注册页面　　　　图 4-5　英文下的注册页面

在上面运行结果中，单击图 4-4 中的“英文”链接则显示图 4-5 中的英文效果注册页面；单击 Chinese 时则再次显示图 4-4 中的运行结果图。

4.4.5　实例分析



源码解析：

在上述实例中，通过使用 Locale 对象来设置语言环境，从而改变注册页面的显示语言。通过使用 Struts 2 中的标签来显示英文和中文的切换超链接，同时使用判断标签来改变选中的超链接样式。

同时通过上面的实例，我们知道通过 struts.locale 属性来设置 Web 应用程序的默认 locale。在程序运行过程中，如果需要修改当前的 locale，只需要提交名为 request_locale 的请求参数即可。如果拦截器从 request_locale 请求参数创建了 Locale 对象，那么就会在 session 中保存一个名为 WW_TRANS_I18N_LOCALE 的属性。

4.5　Struts 2 异常处理

任何成熟的 MVC 框架都应该提供成熟的异常处理机制，Struts 2 也不例外。Struts 2 框架提供了一种声明式的异常处理方式，通过配置拦截器来实现异常处理机制。

视频教学：光盘/videos/04/Exception.avi　　　　长度：7 分钟

4.5.1 基础知识——传统异常处理方式

在传统的 Java 程序中，所有异常对象的根类是 Throwable，从 Throwable 类直接派生的异常类有 Exception 和 Error。



Error 表示 Java 程序中出现一个非常严重的异常错误，这个错误可能是程序所不能够恢复的，例如 ThreadDeath 等。所以说，传统的 Java 异常，主要是指对 Exception 的处理。

在 Exception 异常处理的过程中，一般通过 try{}catch(){}finally 语句，或者使用 throws Exception 来捕获异常。在应用程序中，Exception 异常和从它派生出来的所有异常，都可以通过使用 catch 语句捕捉到。

例如，在本地硬盘 F 盘下创建一个文件夹 exception。在 exception 文件夹中，创建三个 Java 类，在其中的两个类中定义异常信息，然后在第三个 Java 类中引入这两个异常类，从而实现对传统异常的处理。以下是其操作的主要步骤。

(1) 创建两个类文件：MyException.java 和 ThrowableException.java，让这两个类分别继承不同的异常类。MyException 类继承了 Exception 类，该类的主要内容如下所示。

```
public class MyException extends Exception{        //继承 Exception 类
    public MyException(){
        super();
    }
    public MyException(String message){
        super(message);
    }
}
```

在 MyException 类中，实现两个构造方法。如果在调用该类时还有其他参数，可以在该类中定义包含不同参数形式的多个构造方法。

ThrowableException 类继承 Throwable 类，该类的主要内容如下述代码所示。

```
public class ThrowableException extends Throwable{ //继承 Thorwable 类
    public ThrowableException(){
        super();
    }
    public ThrowableException(String message){
        super(message);
    }
}
```



上述两个文件分别继承 Exception 类和 Throwable 类，由于这两个类都是在 Java 的基础包 java.lang 包中，所以不需要使用 import 语句引入。

(2) 在 exception 文件夹下，创建 UseException.java 文件，在该文件中引用前面所定义的异常类。UseException 类的主要内容如下所示。

```
public class UseException {
    public static void myException()throws MyException{
        throw new MyException("MyException 抛出一个异常！");
    }
    public static void throwableException() throws ThrowableException{
        throw new ThrowableException("ThrowableException 抛出一个异常！");
    }
    public static void main(String[] args){
        try{
            UseException.myException();
        }catch(MyException mye){
            System.out.println("Exception:" +mye.getMessage());
            mye.printStackTrace();
        }
        try{
            UseException.throwableException();
        }catch(ThrowableException the){
            System.out.println("Exception:" +the.getMessage());
            the.printStackTrace();
        }
    }
}
```

在 UseException 类中，首先定义 myException()方法和 throwableException()方法，在这两个方法中，分别调用 MyException 类和 ThrowableException 类中包含一个参数的方法。然后在主方法 main()中，使用 try{}catch(){}语句捕获相应的异常，并输出显示。

(3) 编译上述三个 Java 类，然后运行 UseException 类，将输出通过 try{}catch(){}语句捕获到的异常信息和异常路径。输出结果如图 4-6 所示。

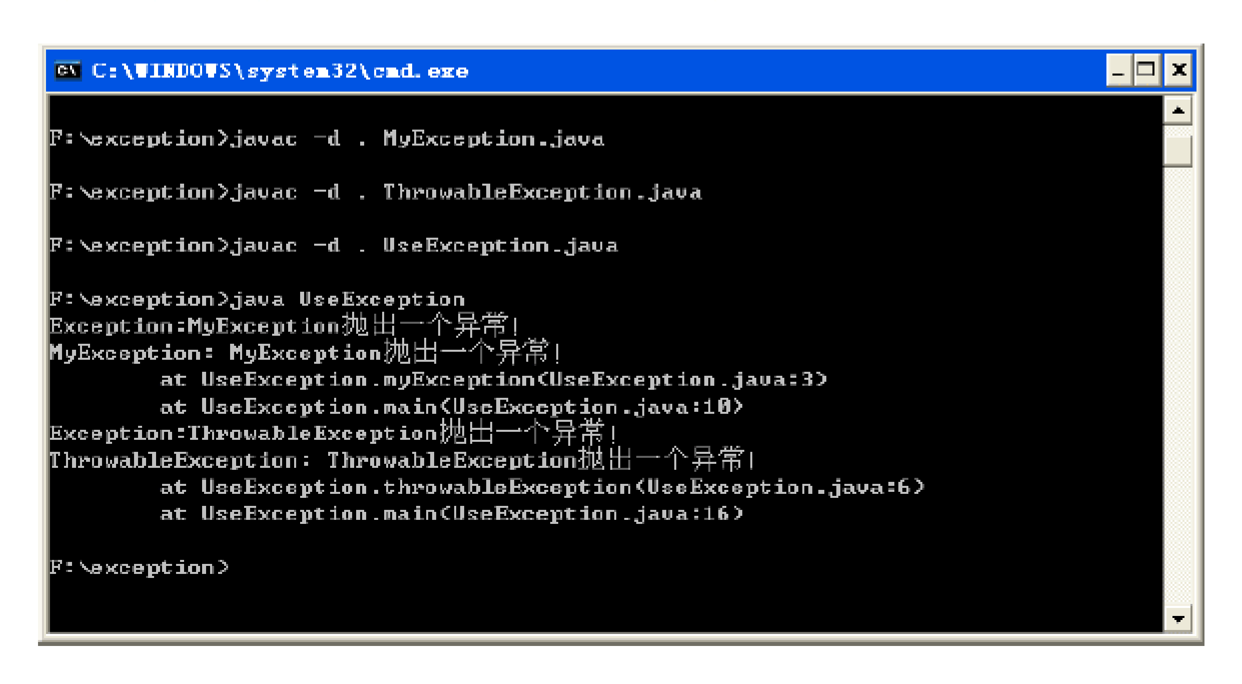


图 4-6　使用传统异常类

4.5.2　基础知识——Struts 2 异常处理机制

在 Struts 2 框架中，可以采用声明式异常处理方式。在这种方式下，只需要在 struts.xml 文件中进行配置，Struts 2 便能够处理异常，然后响应相应的视图，在 Action 中无须编写任何

异常处理代码。



如果 Action 在处理请求的过程中出现异常，一个名为 exception 的拦截器将拦截该异常，并进行处理。

为了实现异常处理，Struts 2 框架在 struts-default.xml 文件中对 exception 拦截器进行了配置，代码如下。

```
<interceptors>
    <interceptor name="exception"
class="com.opensymphony.xwork2.interceptor.ExceptionMappingInterceptor"/>
…
</interceptors>
```

在实现 Struts 2 异常处理时，只需要在 struts.xml 文件中对异常类型和处理异常后返回的结果进行配置就可以了。

如果 Action 在处理请示的过程中出现异常，exception 拦截器将拦截该异常，并根据异常返回到相应的视图页面。Struts 2 框架进行异常处理的流程如图 4-7 所示。

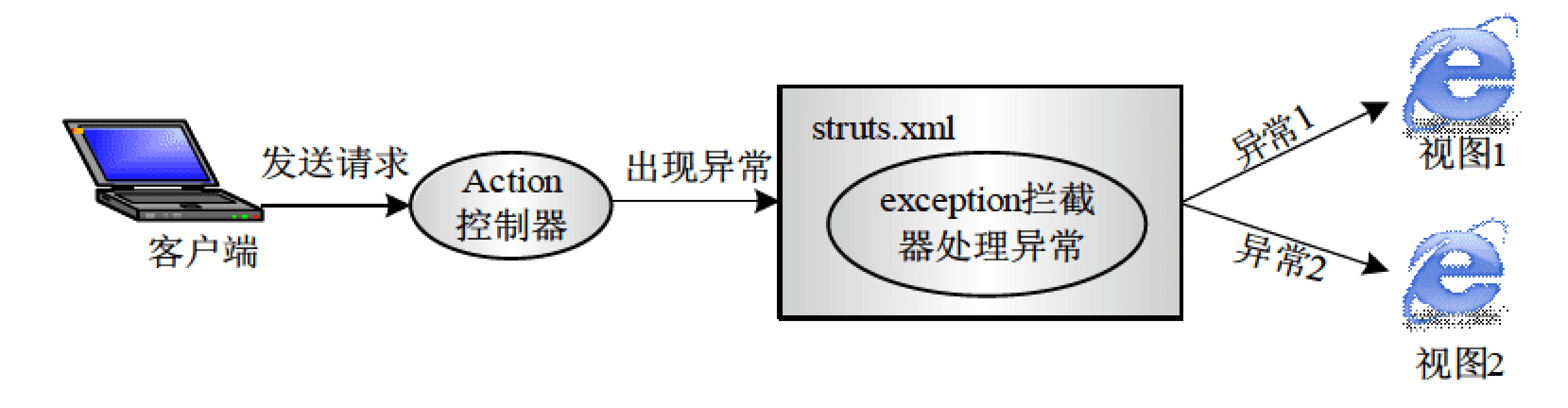


图 4-7　Struts 2 处理异常流程

4.5.3　基础知识——配置异常处理

在本小节中，将介绍配置异常映射的<exception-mapping>元素、对异常配置的分类以及在 JSP 文件中输出异常的方式。

1．配置异常映射

在 struts.xml 文件中，使用<exception-mapping>元素，对 exception 拦截器进行异常映射配置，该元素有以下两个必选属性。

- exception：用来指定出现异常的类型。
- result：用来指定出现异常时，Struts 2 返回给用户的视图名称。

2．异常配置分类

根据异常映射作用的范围，可以将异常映射配置分为以下两类。

1)　全局异常映射

全局异常映射的作用范围是 package 中的所有 Action。这种映射使用<global-exception-mappings>元素进行配置，并在该元素中嵌套<exception-mapping>作为子元素。

2)　局部异常映射

局部异常映射的作用范围是配置元素所在的 Action。这种映射是在 Action 内部，使用<exception-mapping>元素进行配置。



这两种异常映射的优先级不同，当两种映射有冲突时，局部异常映射将覆盖全局异常映射。

例如，在 struts.xml 文件中配置全局异常映射和局部异常映射，代码如下所示。

```
<package name="default" extends="struts-default">
    <global-results>
        <result name="sqlException">/sqlException.jsp</result>
    </global-results>
    <global-exception-mappings>
        <exception-mapping result="sqlException"
exception="java.sql.SQLException"/>        //配置全局异常映射
    </global-exception-mappings>
    <action name="register" class="action.Register">
        <exception-mapping result="myException"
exception="java.lang.Exception"/>      //配置局部异常映射
        <result name="myException">/myException.jsp</result>
    </action>
</package>
```

上述代码中，配置全局异常映射 sqlException，异常类型为 java.sql.SQLException。在全局结果中，为全局映射配置返回结果为 sqlException.jsp。

在<action>元素中，配置局部异常映射 myException，异常类型为 java.lang.Exception，使用<result>元素为 myException 异常进行配置，返回结果为 myException.jsp。

3. 输出异常信息

可以使用 Struts 2 中的 property 标签输出异常信息。

如果输出异常对象本身，代码如下。

```
<s:property value="exception.message"/>
```

如果输出异常堆栈信息，代码如下。

```
<s:property value="exceptionStack"/>
```

4.5.4　实例描述

之前我在刚接触开发的时候经常会遇到这样的一系列问题，比如说编写一个注册程序，当用户在数据年龄中输入的是不合法的字符或者是负数时，程序将会崩溃并抛出异常。学习了 Struts 2 之后，我才发现这样的问题并不难解决。通过使用 Struts 2 异常处理机制就可以实现理想中的效果，下面以一个简单的用户注册抛出异常并且作出相应的处理为实例来进行讲解。

4.5.5 实例应用

【例 4-4】 用户注册异常处理。

本节以用户注册为例，使用 Struts 2 的异常处理机制，捕获相应的异常。首先创建 Web 应用 ch4_exception，配置 Struts 2 开发环境。

(1) 创建 JSP 文件 register.jsp，在该文件中有注册表单，另外定义输出异常信息的语句。register.jsp 文件的主要内容如下所示。

```
<%@ page language="java" pageEncoding="GB2312"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<strong>用户注册</strong><br>
<s:property value="exception.message"/>
<s:form action="register">
    <s:textfield name="userName" label="姓名"/>
    <s:textfield name="userAge" label="年龄"/>
    <s:submit value="提交"/>
</s:form>
```

在 property 标签中，定义 value 属性值为 exception.message，表示输出异常信息。

(2) 在 src/action 文件夹下，新建 Action 类文件 RegisterException.java，在该文件中封装注册信息 userName 和 userAge 属性。RegisterException.java 类的内容如下所示。

```
package action;
import java.sql.SQLException;
import com.opensymphony.xwork2.ActionSupport;
public class RegisterException extends ActionSupport{
    private String userName;
    private int userAge;
    //省略属性的 setXXX()和 getXXX()方法
    public String execute() throws Exception {
        if(userName == null || "".equals(userName)){
            throw new Exception("异常：姓名不能为空！");
        }
        if(userAge < 0 || userAge > 150){
            throw new SQLException("SQL 异常：添加数据异常,年龄不符合要求！");
        }
        if(userName.length() > 10 ){
            throw new SQLException("SQL 异常：添加数据异常,姓名长度不符合要求！");
        }       return SUCCESS;
    }
}
```

在上述 Action 类的 execute()方法中，接收用户输入的姓名和年龄属性值，然后对这些值进行判断，根据不同的情况，定义不同的提示信息。

(3) 在 struts.xml 文件中，配置异常映射和控制器 Action。其中，配置两个异常映射，一个是全局映射，另一个是局部映射。<package>元素中的内容如下所示。

```
<global-results>
    <result name="showException">/register.jsp</result>
</global-results>
<global-exception-mappings>
    <exception-mapping result="showException"
exception="java.lang.Exception"/>
</global-exception-mappings>
<action name="register" class="action.RegisterException">
    <result name="showException">/register.jsp</result>
    <result name="success">/register.jsp</result>
    <exception-mapping result="showException"
exception="java.sql.SQLException"/>
</action>
```

在<global-exception-mappings>全局映射中，配置 java.lang.Exception 异常。在注册 Action 配置中，配置局部异常映射 java.sql.SQLException。

4.5.6　运行结果

运行 Tomcate 服务器启动项目，在地址栏中输入行 http://localhost:8080/ch4_5/regist.action，将用户姓名的内容输入为空，单击【提交】按钮后，显示所定义的异常信息，如图 4-8 所示。

如果用户姓名不为空，将用户年龄输入不符合要求的内容，显示异常信息如图 4-9 所示。

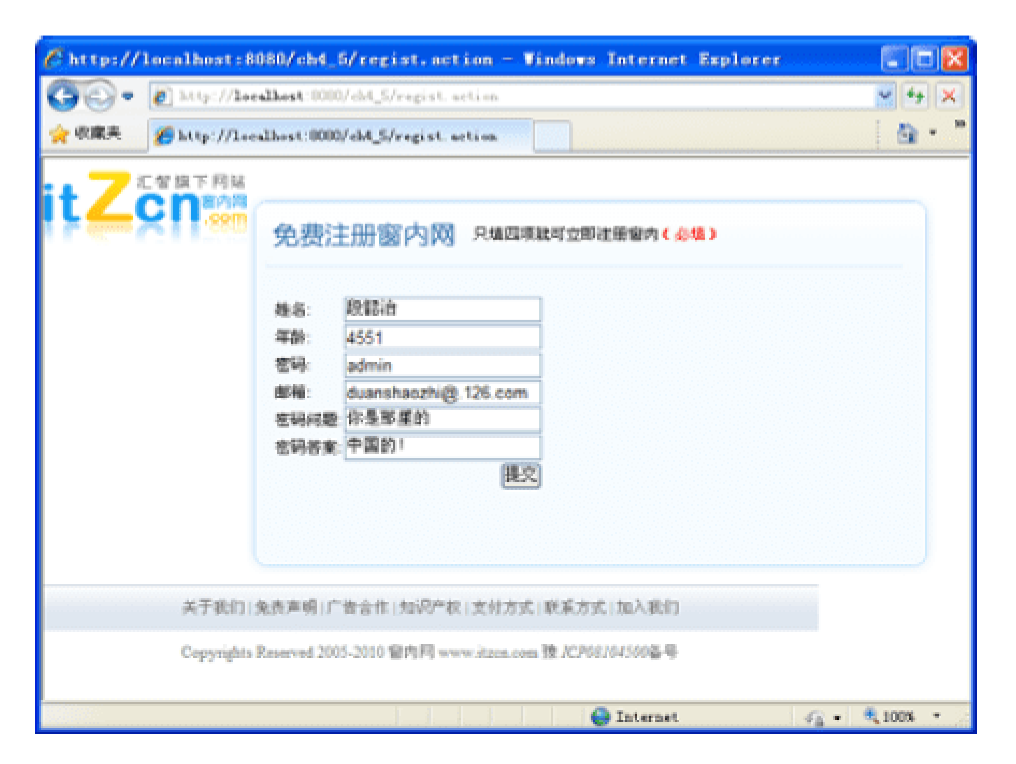


图 4-8　姓名不能为空

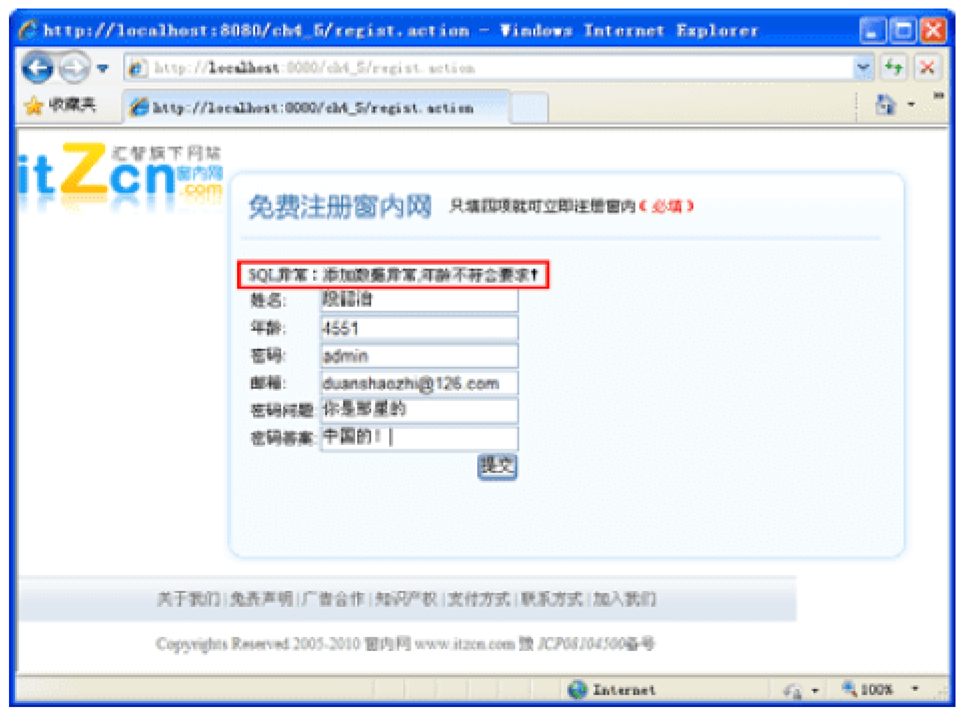


图 4-9　年龄不符合要求

4.5.7　实例分析



源码解析：

上述实例中，在 JSP 页面定义 value 属性值为 exception.message，用来显示错误的消息，RegisterException 类获取提交数据时会判断数据是否合法，如果不合法则抛出异常，然后在 struts.xml 中配置全局和局部的异常映射。

4.6 常见问题解答

4.6.1 Struts 2 国际化中文乱码解决问题



Struts 2 国际化中文乱码如何解决？
网络课堂：http://bbs.itzcn.com/thread-11001-1-1.html

开发工具：MyEclipse，所有的文件编码都设置为 utf-8。所有过程正常开发结束后，将 messagesource.propertis 本地化，代码如下。

```
native2ascii messagesource.propertis messagesource_zh_CN.propertis
```

启动运行，发现页面上从资源文件中读取的内容为乱码。解决方法：本地化时指定编码方式即可，代码如下。

```
native2ascii -encoding UTF-8 messagesource.propertis
messagesource_zh_CN.propertis
```

Java 本身就支持多国语言编码，不需要写任何程序便可以很简单地实现，秘诀就是以下两点。

- 所有 HTML/JSP 页面全部采用 UTF-8 编码。
- 客户端浏览器完全支持 UTF-8 编码。

步骤：

(1) 把所有的 HTML/JSP 的 ContentType 都设为 UTF-8。

(2) 对于 JSP 程序中的非 ASCII 码提示信息，都不应该写在程序里面，应该放在 application.properties 里面统一管理。

(3) 对 HTML 用 native2ascii 工具统一做一次处理，把 HTML 中的非 ASCII 码都转换为 Unicode 编码。

(4) 针对不同的语言，写不同的 application.properties，比如说简体中文是 application_zh_CN.properties，繁体中文是 application_zh_TW.properties，然后对这些配置信息文件同样用 native2ascii 工具处理一次，把非 ASCII 码统一转为 Unicode 编码。

(5) 在 Servlet 类中，使用 request.getCharacterEncoding()获得客户端的操作系统默认编码，然后使用 set()方法将该编码放入 HTTPSession 的 Locale 中。

OK！现在不同的客户访问，就会显示不同的语言版本了。此时自己浏览器的字符集就是 UTF-8。现在网站和 Google 一样了，细心的读者可以发现自己的浏览器访问 Google 的时候也是 UTF-8。

切记：所有的 HTML/JSP 都要设为 UTF-8 编码，所有文件中的非 ASCII 码字符都要用 native2ascii 工具进行转换。

4.6.2　使用 Struts 2 国际化标签的错误问题



使用 Struts 2 国际化标签的错误问题？
网络课堂：http://bbs.itzcn.com/thread-15762-1-1.html

在编写程序时，使用到了<s:url>标签。可是出现了错误，编写的代码如下所示。

```
<%@ page language="java" pageEncoding="GB2312"%>
<html><head>
<meta http-equiv="X-UA-Compatible" content="IE=7">
<meta http-equiv="content-type" content="text/html;charset=gb2312">
<title>国际化</title>
<body>
<s:url id="chinese_url" value="regist.action">
    <s:param name="request locale"
value="@java.util.Locale@CHINA"></s:param>
</s:url>
<s:url id="english_url" value="regist.action">
    <s:param name="request_locale"
value="@java.util.Locale@ENGLISH"></s:param>
</s:url>
</body>
</html>
```

请问是哪里出了问题，我找了好长时间都没有找到原因？

【解决办法】：据我观察之所以出现错误的原因在于没有引入该标签的标签库。你将以下代码引入运行页面应该可以避免出现错误！

```
<%@ taglib prefix="s" uri="/struts-tags" %>
```

上述代码是将 struts-tags 标签库引入到页面中，因为用到的<s:url>标签就在该标签库内。

4.7　习　　题

一、填空题

(1) 国际化的软件应该具有可扩展性、全球化和__________的特征。

(2) 通过软件__________和软件本地化，提供满足特定国际市场的文化、语言和技术要求的产品，使得产品的销售市场扩大。

(3) 在 Java 国际化中，使用 MessageFormat 类处理__________，Struts 2 作为一个优秀的 MVC 框架，对占位符提供更加简单的操作方式。

(4) 在 Struts 2 应用程序中创建资源文件时，如果文件名为 package_language_country.properties，其中 package 为固定的字符串，__________和 country 表示相应的语言和国家，那么该文件就是包范围资源文件。

(5) 在 Struts 2 框架中，允许为每个 Action 创建对应的资源文件，如果该资源文件名称为 RegisterAction_language_country.properties，那么__________为指定 Action 的名称。

(6) 如果不需要客户端修改浏览器的语言首选项，那么在应用程序中，可以通过__________的方式，设置用户的区域语言。

二、选择题

(1) 使用 Java 进行国际化时，所使用的类都是由 java.util 包提供的。该包中不相关的类有__________。

A. UnsupportedClassVersionError
B. ResourceBundle
C. ListResourceBundle
D. PropertyResourceBundle

(2) 下列不属于 Struts 2 框架的资源文件的是__________。

A. 全局资源文件
B. Action 范围资源文件
C. 临时资源文件
D. 永久型资源文件

(3) 在 Struts 2 中，下面__________是不正确的访问位资源文件中的本地化消息内容。

A. 在 JSP 页面中访问本地化消息
B. 在 Action 中访问本地化消息
C. 在表单标签属性中访问本地化消息
D. 在 HTML 中访问国际化消息

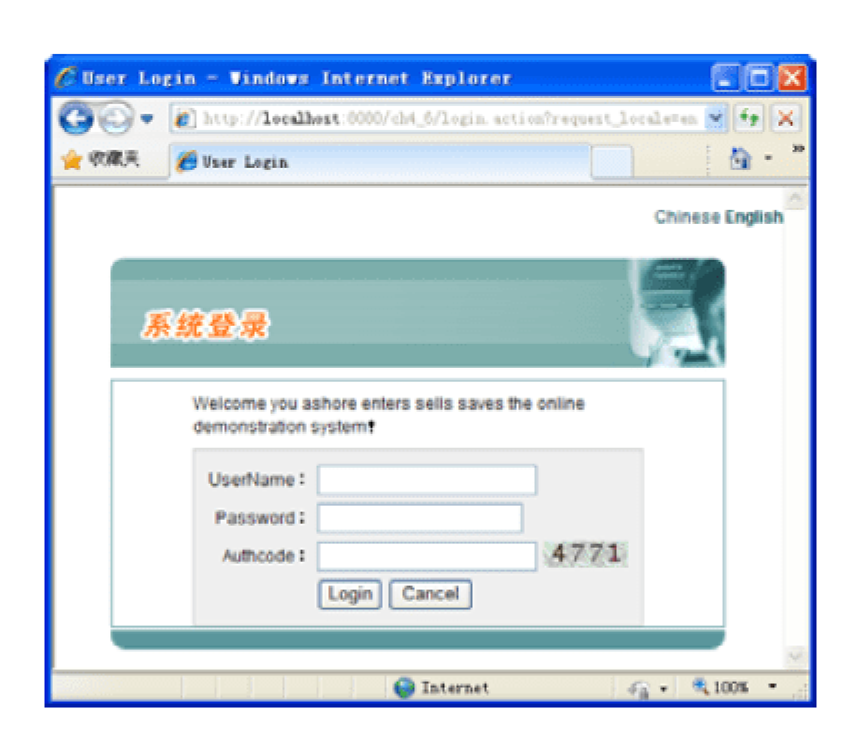
三、上机练习

上机练习：中英文版登录界面。

要求：用户登录界面实现一个简单的中英文切换，当用户单击【英文】按钮时则该页面显示的文字为英文版的登录页面，当用户单击 Chinese 按钮时则显示中文版的登录界面。运行结果如图 4-10 和图 4-11 所示。



图 4-10 中文版登录界面图



4-11 英文版登录界面



第 5 章　Struts 2 中的拦路虎——拦截器

内容摘要：

拦截器(Interceptor)是 Struts 2 的一个重要特性。Struts 2 框架的大多数核心功能都是通过拦截器来实现的，像避免表单重复提交、类型转换、对象组装、验证、文件上传等，都是在拦截器的帮助下实现的。拦截器之所以称为“拦截器”，是因为它可以在 Action 执行之前调用。

Struts 2 允许以一种可插拔的方式来管理 Action 需要完成的通用操作，并将这些通用操作定义为拦截器方法，然后在 struts.xml 文件中配置 Action 时引入该 Action。

这一章将介绍如何配置拦截器(Interceptor)，以及它在 Struts 2 中是如何对 Action 和 JSP 进行拦截的。当然我们也可以自定义拦截器，对其进行拦截。

学习目标：

- 掌握拦截器的配置。
- 掌握自定义拦截器的步骤及配置。
- 熟练运用方法过滤拦截器。
- 了解 Struts 2 的内置拦截器。
- 掌握拦截器注解操作。
- 熟练掌握权限控制拦截器。

5.1　配置和使用拦截器

拦截器(Interceptor)是 Struts 2 的一个强有力的工具，有许多功能都是构建于它之上，如国际化、转换器和校验等。下面将讲解在 Struts 2 中如何配置和使用拦截器。

视频教学：光盘/videos/05/interceptor.avi　　长度：9 分钟
光盘/videos/05/Timer.avi　　长度：4 分钟

5.1.1　基础知识——配置和使用拦截器

拦截器，在 AOP(Aspect Oriented Programming)中用于在某个方法或字段被访问之前，进行拦截。然后在之前或之后加入某些操作。拦截器是 AOP 的一种实现策略。

拦截器允许在 Action 的执行前后插入代码执行。Struts 2 中的拦截器是一个强有力的工具，它可以为 Action 动态添加输入验证(验证用户的输入是否正确)、对象组装(将用户输入的数据转换为对象的属性)、权限控制(确保访问者是登录用户)，日志记录(记录 action 的执行情况)等功能，而不需要修改 Action。

Struts 2 的拦截器实现相对简单。当请求到达 Struts 2 的 ServletDispatcher 时，Struts 2 会查找相对应的配置信息，并实例化拦截器对象。串成一个列表(list)，最后一个一个地调用列表中的拦截器，如图 5-1 所示。

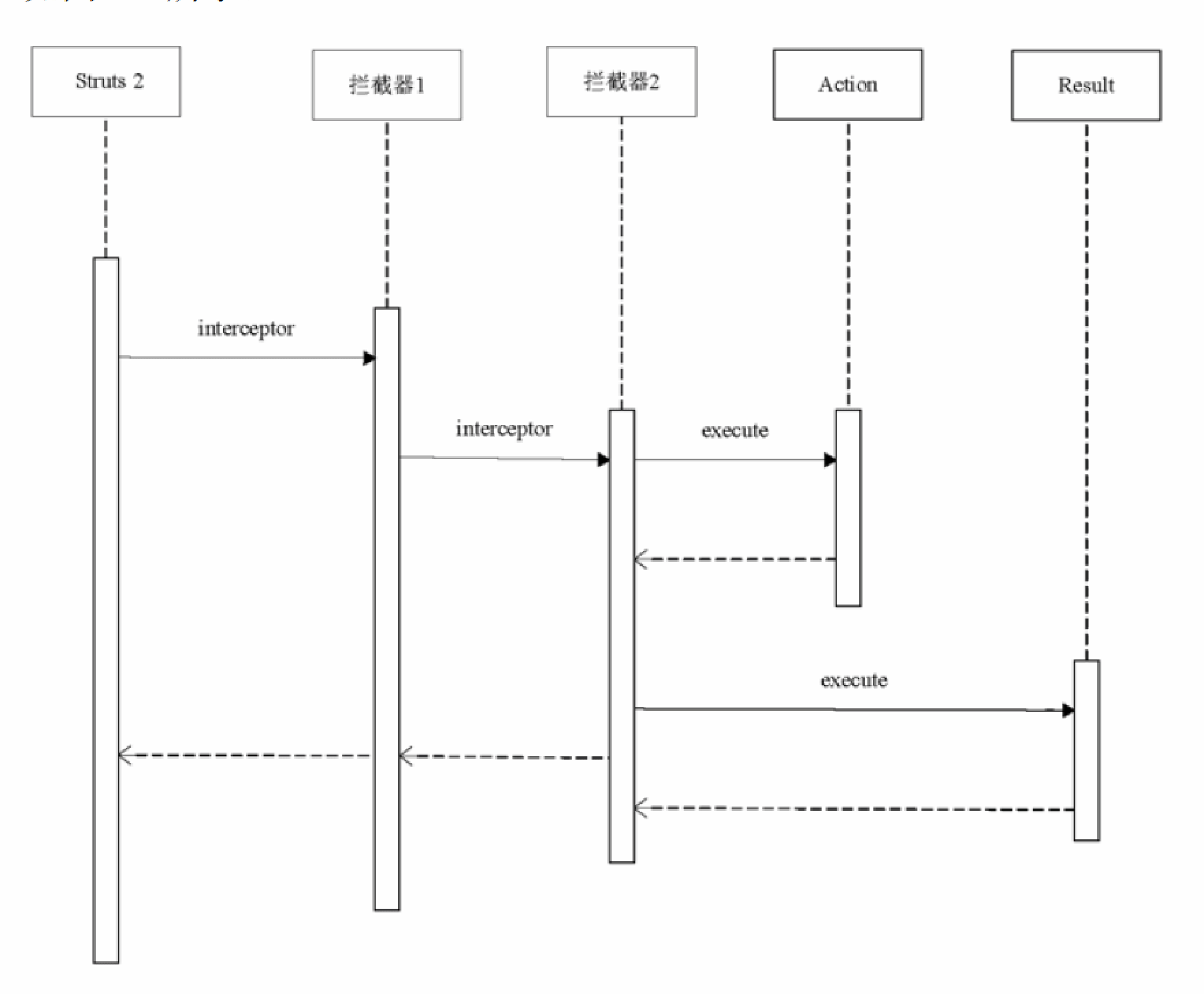


图 5-1　拦截器调用序列图

1. 配置拦截器

引入拦截器机制，可以实现对 Action 通用操作的可插拔式管理，这种可插拔管理是基于 struts.xml 配置文件实现的。

1) 配置拦截器

在 struts.xml 文件中定义拦截器只需为拦截器类指定一个拦截器名称即可。使用<interceptor…/>元素来定义拦截器，其最简单的格式如下。

```
<!-- 通过指定拦截器名和拦截器实现类来定义拦截器 -->
<interceptor name="拦截器名" class="拦截器实现类"/>
```

大部分时候，只需要通过上面的格式就可完成拦截器的配置。如果还需要在配置拦截器时传入拦截器参数，则需要在<interceptor…/>元素中使用<param…/>子元素。下面是在配置拦截器时，同时传入拦截器参数的配置形式。

```
<!-- 通过指定拦截器名和拦截器实现类来定义拦截器 -->
<interceptor name="拦截器名" class="拦截器实现类">
        <!-- 下面元素可以出现 0 次，也可以出现无数次，其中 name 属性指定需要设置的参数名，
             中间指定的就是该参数的值 -->
        <param name="参数名">参数值</param>
</interceptor>
```

2) 配置拦截器栈

除了上面的配置拦截器外，还可以把多个拦截器连在一起组成拦截器栈，例如：如果需要在 Action 执行前同时做登录检查、安全检查和记录日志，则可以把这三个动作对应的拦截器组成一个拦截器栈。定义拦截器栈使用<interceptor-stack…/>元素，拦截器栈是由多个拦截器组成的，因此需要在<interceptor-stack…/>元素中使用<interceptor-ref…/>元素来定义多个拦截器引用，即该拦截器栈由多个<interceptor-ref…/>元素指定的拦截器组成。



从程序结构上来看，拦截器栈是由多个拦截器组成的，即一个拦截器栈包含了多个拦截器；但从程序功能上来看，拦截器栈和拦截器是统一的，它们包含的方法都会在 Action 的 execute 方法执行之前自动执行。

下面我们就来配置一下拦截器栈，代码如下所示。

```
<interceptor-stack name="拦截器栈">
        <interceptor-ref name="拦截器一"/>
        <interceptor-ref name="拦截器二"/>
        <!-- 还可配置更多的拦截器 -->
        …
</interceptor-stack>
```

上面的配置片段示例，配置了一个名为"拦截器栈"的拦截器栈，这个拦截器由下面的"拦截器一"和"拦截器二"组成，当然还可以包含更多的拦截器，只需在<interceptor-stack…/>元素下配置更多的<interceptor-ref…/>子元素即可。



一旦将上面的“拦截器一”和“拦截器二”配置成了名为“拦截器栈名”的拦截器栈后，就可以完全像使用普通拦截器一样使用拦截器栈，因为拦截器和拦截器栈的功能是完全相同的。

由于拦截器栈与拦截器的功能几乎完全相同，因此可能出现的情况是：拦截器栈里再次包含拦截栈。因此，可能出现如下的配置片段如下。

```
<interceptor-stack name="拦截器栈一">
        <interceptor-ref name="拦截器一"/>
        <interceptor-ref name="拦截器二"/>
        <!-- 还可配置更多的拦截器 -->
        …
</interceptor-stack>
<interceptor-stack name="拦截器栈二">
        <interceptor-ref name="拦截器三"/>
        <interceptor-ref name="拦截器栈一"/>
        <!-- 还可配置更多的拦截器 -->
        …
</interceptor-stack>
```

在上面的配置片段中，第二个拦截器包含了第一个拦截器栈(包含两个拦截器)，其实质就是拦截器栈二由三个拦截器组成：拦截器三、拦截器一和拦截器二。

读者可能会问：为什么不直接在拦截器栈二中直接配置三个拦截器的引用，而是通过引用另一个拦截器栈呢？答案是为了软件复用。因为系统中已经存在了一个名为“拦截器栈一”的拦截器栈，这个拦截器栈由两个拦截器组成，当需要再次使用这两个拦截器时，直接调用该拦截器栈即可，无须分别调用两个拦截器。而且，这两个拦截器组成的拦截器栈也可以单独使用。

在使用拦截器时指定参数值，应通过<interceptor-ref…/>元素增加<param…/>子元素为拦截器指定参数值。

下面是在配置拦截器栈时为拦截器动态指定参数值的语法。

```
<interceptor-stack name="拦截器栈一">
        <interceptor-ref name="拦截器一">
                <!-- 下面为拦截器分别定义了两个参数值 -->
                <param name="参数一">参数值一</param>
                <param name="参数二">参数值二</param>
                <!-- 还可指定更多的参数值 -->
                …
        </interceptor-ref>
        <interceptor-ref name="拦截器二"/>
        <!-- 还可配置更多的拦截器 -->
        …
</interceptor-stack>
```



有两个时机为拦截器指定参数值，一个时机是当定义拦截器(通过<interceptor…/>元素来定义拦截器)时指定拦截器的参数值；另一个时机是当使用拦截器(通过<interceptor-ref…/>元素使用拦截器)时指定拦截器的参数值。前者指定的是拦截器参数的默认值。

如果在两个时机为同一个参数指定了不同的参数值，则使用拦截器时指定的参数将会覆盖默认的参数值。

2. 使用拦截器

一旦定义了拦截器和拦截器栈后，就可以使用拦截器或拦截器栈来拦截 Action 了，拦截器(包含拦截器栈)的拦截行为会在 Action 的 execute()方法执行之前被执行。

1)　为 Action 配置拦截器

通过<interceptor-ref.../>元素可以在 Action 内使用拦截器，在 Action 中使用拦截器的配置语法与配置拦截器栈时引用特定拦截器的语法完全一样。

下面是在 Action 中定义拦截器的配置示例。

```
<include file ="struts-default.xml" />
    <package name="intercep" namespace="/intercep" extends="struts-default">
        <interceptors>
            <!-- 定义第一个拦截器 -->
            <interceptor name="myFirstInterceptor"
class="com.myfirst.interceptor.MyFirstInterceptor"/>
            <!-- 定义第二个拦截器 -->
            <interceptor name="mySecondInterceptor"
class="com.mysecond.interceptor.MySecondInterceptor">
                    <!-- 指定该拦截器的默认参数值 -->
                    <param name="name">第二个拦截器</param>
            </interceptor>
    </interceptors>
    <!-- 配置自定义的 Action，实现类为 com.myinterceptor.LoginAction -->
    <action name="login" class="com.myinterceptor.LoginAction">
        <!-- 配置该 Action 的两个局部结果 -->
        <result name="error">/error.jsp</result>
        <result name="success">/success.jsp</result>
        <!-- 拦截器一般配置在 result 元素之后 -->
        <interceptor-ref name="defaultStack"/>
        <!-- 使用第一个拦截器 -->
        <interceptor-ref name="myFirstInterceptor"/>
        <!-- 使用第二个拦截器 -->
        <interceptor-ref name="mySecondInterceptor">
            <!-- 为该 Action 动态指定拦截器参数 -->
            <param name="name">动态参数</param>
        </interceptor-ref>
    </action>
    </package>
```

如果在 Action 中配置多个拦截器，则会按照引用拦截器的顺序执行。

2)　为 Action 配置拦截器栈

如果一个 Action 需要多个拦截器，引用它们是一件繁琐的事。在此可以将多个拦截器组合

在一起，组成一个拦截器栈，然后在 Action 中直接引用拦截器栈。

```
<package name="intercep" namespace="/intercep" extends="struts-default">
    <interceptors>
            <!-- 定义第一个拦截器 -->
            <interceptor name="myFirstInterceptor"
class="com.myfirst.interceptor.MyFirstInterceptor"/>
            <!-- 定义第二个拦截器 -->
            <interceptor name="mySecondInterceptor"
class="com.mysecond.interceptor.MySecondInterceptor">
                    <!-- 指定该拦截器的默认参数值 -->
                    <param name="name">第二个拦截器</param>
            </interceptor>
            <!-- 定义一个拦截器栈 -->
            <interceptor-stack name="myInterceptorStack">
                    <interceptor-ref name="myFirstInterceptor"/>
                    <interceptor-ref name="mySecondInterceptor"/>
            </interceptor-stack>
    </interceptors>
    <!-- 配置自定义的 Action，实现类为 com.myinterceptor.LoginAction -->
    <action name="login" class="com.myinterceptor.LoginAction">
        <!-- 配置该 Action 的两个局部结果 -->
        <result name="error">/error.jsp</result>
        <result name="success">/success.jsp</result>
        <!-- 拦截器一般配置在 result 元素之后 -->
        <interceptor-ref name="defaultStack"/>
        <!-- 为 Action 配置拦截器栈 -->
        <interceptor-ref name="myInterceptorStack"/>
    </action>
    </package>
```

如果多个 Action 都需要引用相同的拦截器，可以使用 default-interceptor-ref 元素来定义一个默认的拦截器或拦截器栈引用，这样就不需要为每个 Action 指定引用信息了。

例如下面的代码定义了默认的拦截器栈引用。

```
<default-interceptor-ref name="myInterceptorStack"/>
```



如果在一个 Action 中定义了其他的拦截器引用，那么这个 Action 将不再使用默认的拦截器引用。如果 Action 想要在默认拦截器引用的基础上添加新的拦截器，那么只能在 Action 中重新配置默认拦截器引用中的拦截器。

Struts 2 提供了丰富多彩的、功能齐全的拦截器，读者可以到 Struts 2-all-x.jar 或 struts 2-core-2.1.8.jar 包中的 struts-default.xml 中查看关于默认的拦截器与拦截器链的配置，这里就不再多说了。

5.1.2　实例描述

当程序非常大时，启动服务器并将该程序布置到服务器上需要花费很长的时间，甚至达到几小时。而到底用了多长时间成为程序员心中的困惑，那么下面就来做一个可以计算出程序的耗时功能吧！

5.1.3　实例应用

【例 5-1】 测试程序的耗时。

(1) 创建 Action(TimerInterceptorAction.java)，内容如下。

```
package com.struts2.actions;
import com.opensymphony.xwork2.ActionSupport;
public class TimerInterceptorAction extends ActionSupport {
@Override
    public String execute(){
        try  {
            // 模拟耗时的操作
            Thread.sleep( 500 );
        } catch  (Exception e)  {
            e.printStackTrace();
        }
        return SUCCESS;
    }
}
```

(2) 在 src 目录下的 struts-config 文件夹中新建 timer.xml，对 Action 进行配置，代码如下。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
<include file ="struts-default.xml" />
    <package name ="InterceptorDemo" extends ="struts-default" >
        <action name ="Timer" class
="com.struts2.actions.TimerInterceptorAction" >
            <interceptor-ref name ="timer" />
            <result>/Timer.jsp </result>
        </action >
    </package >
</struts>
```

(3) 把 timer.xml 文件引入到 struts.xml 中。

5.1.4 运行结果

运行 Timer.action，查看服务器的后台输出，结果如图 5-2 所示。

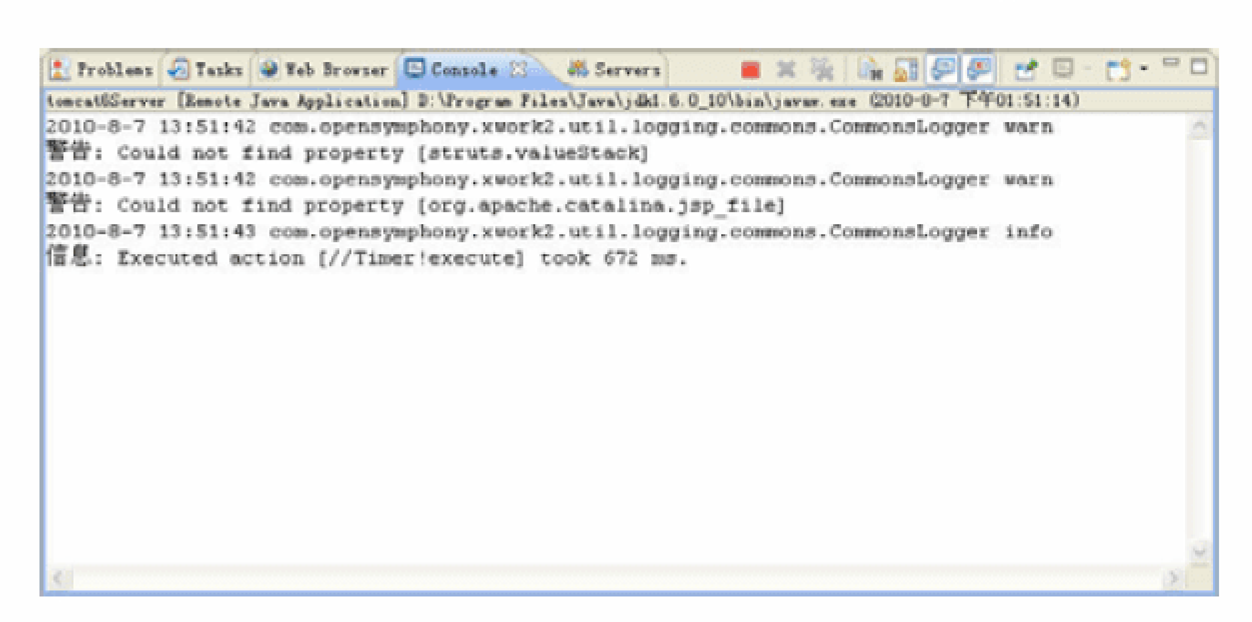


图 5-2　使用Struts 2 框架拦截器timer后台服务器的输出结果

5.1.5 实例分析



源码解析：

上述实例中，在 struts-default.xml 中已经配置了好多的拦截器。如果读者想要使用，只需要在应用程序 struts.xml 文件中通过“<include file="struts-default.xml"/>”将 struts-default.xml 文件包含进来，并继承其中的 struts-default 包(package)，最后在定义 Action 时，使用“<interceptor-ref name="xx"/>”引用拦截器或拦截器栈(interceptor stack)即可。一旦继承了 struts-default 包，所有 Acton 都会调用拦截器栈——defaultStack。当然，在 Action 配置中加入“<interceptor-ref name="xx"/>”可以覆盖 defaultStack。

5.2 自定义拦截器

提到“自定义”这个词，我想读者一定会惊喜 Struts 2 的出现，它就像是坐公交车不如坐自家的小轿车一样简单、快捷、方便。

 视频教学：光盘/videos/05/MyInterceptor.avi　　长度：10 分钟

5.2.1 基础知识——自定义拦截器

Struts 2 框架提供了许多内置的拦截器，这些内置的拦截器实现了 Struts 2 的大部分功能，因此，大部分 Web 应用的通用功能，都可以通过直接使用这些拦截器来完成。还有一些系统逻辑相关的通用功能，则可以通过自定义拦截器来实现。

1. 实现自定义拦截器类

如果用户要开发自己的拦截器类，应该实现 com.opensymphony.xwork2.interceptor. Interceptor 接口，该接口的类定义代码如下。

```
import java.io.Serializable;
import com.opensymphony.xwork2.ActionInvocation;
public interface Interceptor extends Serializable {
    //销毁该拦截器之前的回调方法
    void destroy();
    //初始化该拦截器的回调方法
    void init();
    //拦截器实现拦截的逻辑方法
    String intercept(ActionInvocation invocation) throws Exception;
}
```

通过上面的接口可以看出，该接口里包含了三个方法。

- init()：拦截器被初始化之后，在该拦截器执行拦截之前，系统将回调该方法。对于每个拦截器而言，该 init()方法只执行一次。因此，该方法的方法体主要用于打开一些一次性资源，例如数据库资源等。
- destroy()：该方法与 init()方法对应。在拦截实例被销毁之前，系统将回调该拦截器的 destroy()方法，该方法用于销毁在 init()方法里打开的资源。
- intercept(ActionInvocation invocation)：该方法是用户需要实现的拦截动作。就像 Action 的 execute()方法一样，intercept()方法会返回一个字符串作为逻辑视图。如果该方法直接返回了一个字符串，系统将会跳转到该逻辑视图对应的实际视图资源，不会调用被拦截的 Action。该方法的 ActionInvocation 参数包含了被拦截的 Action 的引用，可以通过调用该参数的 invoke()方法，将控制权转给下一个拦截器，或者转给 Action 的 execute()方法。

除此之外，Struts 2 还提供了一个 AbstractInterceptor 类，该类提供了一个 init()和 destroy()方法的空实现，如果实现的拦截器不需要申请资源，则可以无须实现这两个方法。可见，采用继承 AbstractInterceptor 类来实现自定义拦截器会更加简单。



当实现 intercept(ActionInvocation invocation)方法时，可以获得 ActionInvocation 参数，这个参数又可以获得被拦截的 Action 实例，一旦取得了 Action 实例，几乎获得了全部的控制权：可以实现将 HTTP 请求中的参数解析出来，设置成 Action 的属性(这是系统 params 拦截器干的事情)；也可以直接将 HTTP 请求中的 HttpServletRequest 实例和 HttpServletResponse 实例传给 Action(这是 servlet-config 拦截器干的事情)……当然，也可实现应用相关的逻辑。

2. 使用自定义拦截器

使用自定义拦截器需要两个步骤。

(1) 通过<interceptor.../>元素来定义拦截器。

(2) 通过<interceptor-ref.../>元素来使用拦截器。

为了给读者以清晰的讲解，对于上面自定义的拦截器类，其配置代码如下。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- 指定拦截器的 DTD 信息 -->
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <!-- 通过常量配置 Struts 2 所使用的解码集 -->
    <constant name="struts.i18n.encoding" value="gbk" />
    <constant name="struts.devMode" value="true" />
    <include file ="struts-default.xml" />
    <package name="intercep" namespace="/intercep" extends="struts-default">
        <!-- 配置应用所用的拦截器 -->
        <interceptors>
                <!-- 配置 mysimple 拦截器 -->
                <interceptor name="mysimple" class="SimpleInterceptor">
                    <!-- 为拦截器指定参数值 -->
                    <param name="name">简单拦截器</param>
                </interceptor>
        </interceptors>
        <!-- 配置自定义的 Action，实现类为 com.myinterceptor.LoginAction -->
        <action name="login" class="com.myinterceptor.LoginAction">
            <!-- 配置该 Action 的两个局部结果 -->
            <result name="error">/error.jsp</result>
            <result name="success">/success.jsp</result>
            <!-- 拦截器一般配置在 result 元素之后 -->
            <interceptor-ref name="defaultStack"/>
            <!-- 为 Action 配置拦截器 -->
            <interceptor-ref name="mysimple">
                <param name="name">改名后的拦截器</param>
            </interceptor-ref>
        </action>
    </package>
</struts>
```

通过上面的配置文件，将 SimpleInterceptor 拦截器类定义成名为“mysimple”的拦截器，并在名为 login 的 Action 中使用该拦截器。值得注意的是，当配置 mysimple 拦截器时，需要手动引用系统默认的 defaultStack 拦截器栈。



前面已经讲过，如果为 Action 指定一个拦截器，则系统默认的拦截器栈将会失去作用。为了继续使用默认拦截器，所以上面的配置文件中手动引入了默认的拦截器。

5.2.2　实例描述

当我们做了一个后台管理系统后，并且该管理系统为一个企业的内部系统，那么是否每个人都可以登录，都可以进入到管理系统的主页面呢？那么是否可以对该系统进行操作呢？这样后果不堪设想，因此一个程序的“门”至关重要，用于拦截非法登录的用户。

5.2.3　实例应用

【例 5-2】 为程序做扇“门”。

(1)　编辑 LoginAction 类，代码如下。

```
package com.struts2.actions;
import com.opensymphony.xwork2.ActionSupport;
public class LoginAction extends ActionSupport {
    private static final long serialVersionUID = -4278369739741451645L;
    private String username;//用户名
    private String pass;//密码
    @Override
    public String execute() throws Exception {
        return "main";
    }
    /*下面是 username 和 pass 的 get、set 方法，在此省略了*/
    …
}
```

(2)　接着编辑拦截 LoginAction 的拦截器类，判断用户是否合法，代码如下。

```
package com.struts2.interceptor;
import java.util.Date;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts2.ServletActionContext;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.AbstractInterceptor;
import com.struts2.actions.LoginAction;
public class SimpleInterceptor extends AbstractInterceptor {
    //定义拦截器名称
    private String name;
    public void setName(String name)
    {
        this.name = name;
    }
    public String intercept(ActionInvocation invocation)throws Exception
    {
        //获取 HttpServletRequest 对象
        HttpServletRequest request=ServletActionContext.getRequest();
        //获取被拦截的 Action 实例
```

```
        LoginAction action = (LoginAction)invocation.getAction();
        System.out.println(name+"拦截器的动作:开始执行登录 Action 的时间为: "+new
Date());
        //把得到的 username 赋给 LoginAction 中的 username 属性
        action.setUsername(request.getParameter("username"));
        action.setPass(request.getParameter("pass"));
        //判断用户是否是合法用户
        if(!action.getUsername().equals("admin")||!action.getPass().equals
        ("admin"))
        {
            System.out.println("user name error");
            return "input";
        }
        //获取开始时间
        long start = System.currentTimeMillis();
        String result = invocation.invoke();
        System.out.println(name+"拦截器的动作: 执行完登录 Action 的时间为: " + new
Date());
        //获取结束时间
        long end = System.currentTimeMillis();
        System.out.println("拦截器的动作: 执行 action 事件的时间是:"+(end-start)+"
毫秒");
        return result;
    }
}
```

(3) 在 src 下的 struts-config 文件夹中新建 login.xml，配置 Action 和 Action 的拦截器，代码如下。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name ="login" extends ="struts-default" >
        <interceptors>
            <interceptor name="loginInterceptor"
class="com.struts2.interceptor.SimpleInterceptor">
                    <!-- 为拦截器指定参数值 -->
                    <param name="name">简单拦截器</param>
            </interceptor>
        </interceptors>
        <action name ="login" class ="com.struts2.actions.LoginAction" >
            <result name="main">/main.jsp</result>
            <result name="input">/login.jsp</result>
            <!-- 在 Action 中引用拦截器 -->
            <interceptor-ref name="loginInterceptor">
                <param name="name">改名后的拦截器</param>
```

```
            </interceptor-ref>
        </action >
    </package >
</struts>
```

(4) 编辑登录页面 login.jsp，代码如下。

```
<form name=form1 action="login/login.action" method=post>
        <input type="text" name="username"/>
        <input type="text" name="pass"/>
        <input type="submit" value=" 登 录 "/>
</form>
```

(5) 编辑登录成功页面 main.jsp。

5.2.4 运行结果

运行 login.jsp 页面，如果输入的密码不是“admin”，单击“登录”按钮返回本页面，如图 5-3 所示。如果输入的用户名和密码都是“admin”，登录成功，跳转至 main.jsp 页面，如图 5-4 所示。



图 5-3　登录失败，返回登录页面



图 5-4　登录成功，跳转至主页面

不要光顾着表面工作，我们再来看看控制台输出，如图 5-5 所示。

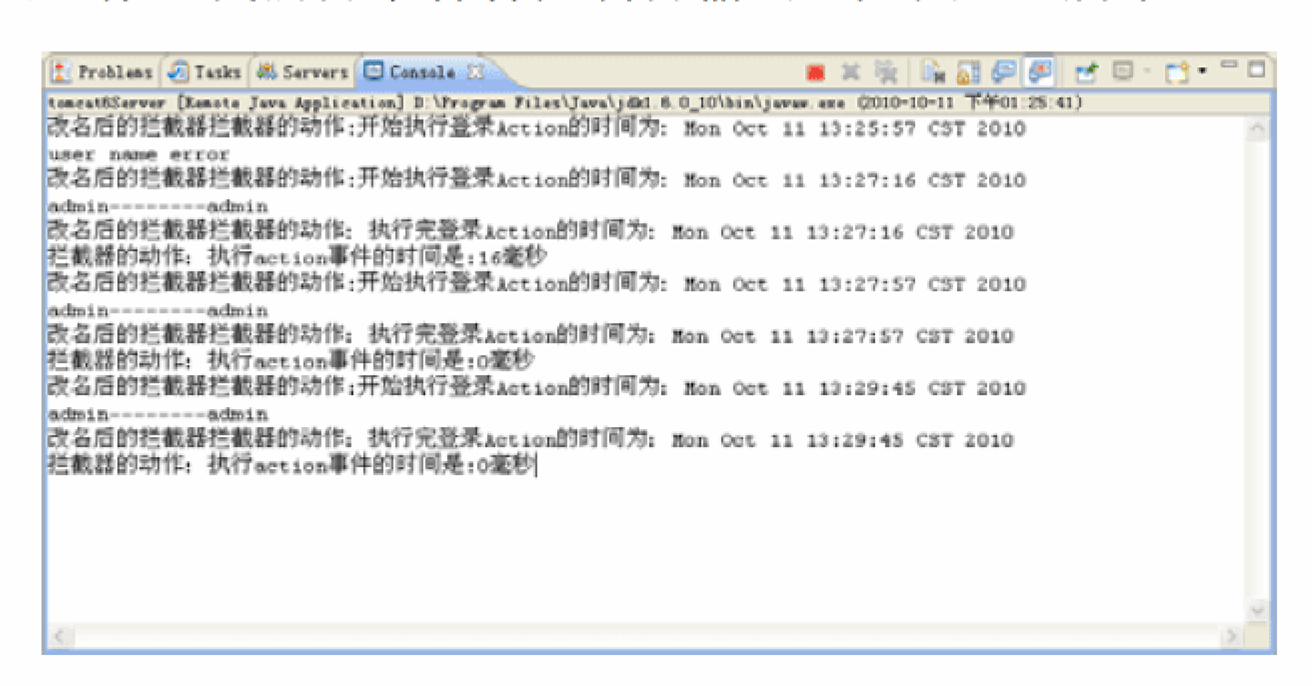
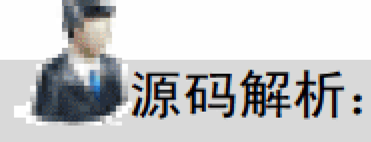


图 5-5　控制台输出

5.2.5　实例分析



源码解析：

在上述案例中，在拦截器类 SimpleInterceptor 中给 LoginAction 类的两个属性 username 和 pass 赋值，读者可能会问在拦截器类中既然已经得到了要拦截的 Action 类 LoginAction，不就是得到了它的两个属性了吗？答案是错的。在这里根本得不到其属性，因为拦截器是在执行 Action 之前执行的，在此之前 LoginAction 未执行，因此要想在拦截器类中得到页面中的元素，需要用 request.getParameter()方法来获取。

5.3　拦截器深度剖析

前面介绍了 Struts 2 拦截器的原理、配置和简单使用，本节将对拦截器进行深入的探讨。

视频教学： 光盘/videos/05/FunFilter.avi　　长度：10 分钟
光盘/videos/05/PreResultListener.avi　　长度：5 分钟
光盘/videos/05/FunFilter.avi　　长度：10 分钟

5.3.1　基础知识——深度剖析拦截器

拦截器的配置离不开前面介绍的<interceptor.../>、<interceptor-ref.../>和<interceptor-stack.../>元素。但对于深入拦截器的配置方面，还有一些值得注意的地方。

1．方法过滤

在默认情况下，如果为某个 Action 定义了拦截器，则这个拦截器会拦截该 Action 内的所有方法。在有些情况下，也许无需拦截所有的方法，只需要拦截某些方法，此时就需要使用 Struts 2 拦截器的方法过滤特性。

为了实现方法过滤的特性，Struts 2 提供了一个 MethodFilterInterceptor 类，该类是 AbstractInterceptor 类的子类，如果用户需要自己实现的拦截器支持方法过滤特性，则应该继承 MethodFilterInterceptor 类。该类需要实现一个方法(在 MethodFilterInterceptor.java 类中的方法)。

```
protected String doIntercept(ActionInvocation ai) throws Exception {
        // TODO Auto-generated method stub
        return null;
    }
```

从上面的代码可以看出，拦截器的拦截逻辑与前面介绍的简单拦截器的拦截逻辑相同，只是之前需要重写 intercept()方法，现在需要重写 doIntercept()方法。

实际上，实现方法过滤的拦截器与实现普通拦截器并没有太大的区别，只需要注意两个方面：实现方法过滤的拦截器需要继承 MethodFilterInterceptor 抽象类，并且重写 doIntercept()方法对 Action 定义的拦截逻辑。

在 MethodFilterInterceptor()方法中，额外增加了如下两个方法。

- public void setExcludeMethods(String excludeMethods)：排除需要过滤的方法——设置方法“黑名单”，所有在 excludeMethods 字符串中列出的方法都不会被拦截。
- public void setIncludeMethods(String includeMethods)：设置需要过滤的方法——设置方法“白名单”，所有在 includeMethods 字符串中列出的方法都会被拦截。



如果一个方法同时在 excludeMethods 和 includeMethods 中列出，则该方法会被拦截。

因为 MethodFilterInterceptor 类包含了如上两个方法，所以该拦截器的子类也会获得这两个方法。可以在配置文件中指定需要被拦截或者不需要被拦截的方法。

```
<interceptors>
    <interceptor name="拦截器名" class="拦截器类"/>
</interceptors>
<action name="Action 名" class="Action 类">
    <interceptor-ref name="拦截器名">
    <!-- 指定 execute 方法不需要被拦截 -->
        <param name="excludeMethods">execute</param>
    </interceptor-ref>
</action>
```

在上面的配置文件中，通过 excludeMethods 属性指定了 execute()方法，无须被拦截。

如果需要同时指定多个方法不被拦截器拦截，则多个方法之间以英文逗号(,)隔开。如下所示。

```
<interceptor-ref name="要引用的拦截器名">
        <!-- 指定方法一、方法二、方法三不需要被拦截 -->
        <param name="excludeMethods">方法一,方法二,方法三</param>
</interceptor-ref>
```

如果 excludeMethods 参数和 includeMethods 参数同时指定了一个方法名，则拦截器会拦截该方法，如下所示。

```
<interceptor-ref name="要引用的拦截器名">
        <!-- 指定方法一、方法二、方法三不需要被拦截 -->
        <param name="excludeMethods">方法一,方法二,方法三</param>
        <!-- 指定方法一需要被拦截 -->
        <param name="includeMethods">方法一</param>
</interceptor-ref>
```



上面配置中通过 excludeMethods 参数指定了方法一、方法二和方法三不需要被拦截，又通过 includeMethods 参数指定了方法一需要拦截——二者冲突以 include Methods 参数指定的取胜，即拦截器会拦截方法一。

2. 拦截监听的结果

在前面的简单拦截器中，我们将在 execute()方法执行之前、执行之后的动作都定义在拦截器的 intercepte(ActionInvocation invocation)方法中，这种方式使结构看上去不够清晰。

为了精确定义在 execute()方法执行结束后再处理 Result 执行的动作，Struts 2 提供了用于拦截结构的监听器，这个监听器是通过手工注册到拦截器内部的。

实现拦截结构的监听器必须实现 PreResultListener 接口，其代码如下。

```
public class MyPreResultListener implements PreResultListener {
    //定义在处理 Result 之前的行为
    public void beforeResult(ActionInvocation invocation,String resultCode){
        //打印出执行结果
        System.out.println("最后的逻辑视图为"+resultCode);
    }
}
```

与前面直接在 intercepte 方法中定义的，在 execute()方法执行之后执行的代码相比，在 beforeResult()方法内有了另外一个参数：resultCode，这个参数就是被拦截 Action 的 execute()方法的返回值。上面的监听器仅仅打印了被拦截的 Action 中的 execute()方法的返回值。

虽然上面的 beforeResult()方法也获得 ActionInvocation 类型的参数，但通过这个参数来控制 Action 的作用已经不再那么明显——因为 Action 的 execute()方法已经执行完了。

监听器需要通过代码手动注册给某个拦截器，需要在拦截器类中的 intercept (ActionInvocation invocation)方法中加上下面代码进行注册监听器。

```
invocation.addPreResultListener(new MyPreResultListener());
```

通过代码手动注册了一个拦截结果的监听器 MyPreResultListener，该监听器中的 before Result()方法肯定会在系统处理 Result 之前执行。

虽然在 beforeResult()方法中再对 ActionInvocation 方法处理已经没有太大的作用，但是 Action 已经执行完毕，而 Result 还没有开始执行的时间点也非常重要，可以让开发者精确控制到 Action 处理的结果，并且对处理结果做出针对性的处理。



虽然 beforeResult()方法中也可获得 ActionInvocation 实例，但不能通过该实例再次调用 invoke()方法，如果再次调用 invoke()方法，将会再次执行 Action 处理，Action 处理之后紧跟的是 beforeResult()方法……这会陷入一个死循环。

3．覆盖拦截器中特定拦截器的参数

有些时候，Action 需要引用一个拦截器栈，当引用这个拦截器栈时，又需要覆盖这个拦截器栈中某个拦截器的特定参数。

```
<interceptors>
    <!-- 配置第一个拦截器 -->
    <interceptor name="拦截器一" class="拦截器一类">
        <!-- 为拦截器指定参数值 -->
        <param name="name">拦截器一</param>
    </interceptor>
    <!-- 配置第二个拦截器 -->
    <interceptor name="拦截器二" class="拦截器二类"/>
    <!-- 配置拦截器栈 -->
    <interceptor-stack name="拦截器栈">
        <!-- 配置拦截器战中的第一个拦截器 -->
        <interceptor-ref name="拦截器一">
            <param name="name">第一个</param>
        </interceptor-ref>
        <!-- 配置拦截器栈中的第二个拦截器 -->
        <interceptor-ref name="拦截器二">
            <param name="name">第二个</param>
        </interceptor-ref>
    </interceptor-stack>
</interceptors>
<action name="Action 名" class="Action 类">
    <!-- 引用上面的拦截器栈 -->
    <interceptor-ref name="拦截器栈"/>
</action>
```

在上面的代码中，配置了一个名为“拦截器栈”的拦截器栈，这个拦截器栈包含两个拦截器，这两个拦截器分别引用“拦截器一”和“拦截器二”的拦截器，并且覆盖了“拦截器一”的默认参数。在 Action 中使用拦截器栈时，并未覆盖拦截器栈中的任何拦截器的参数。

如果需要覆盖“拦截器二”中的 name 属性值，最容易的方法就是在引用拦截器栈的<interceptor-ref.../>元素中增加<param.../>子元素，即将配置 Action 的配置片段修改成如下形式。

```
<action name="Action 名" class="Action 类">
    <!-- 引用上面的拦截器栈 -->
    <interceptor-ref name="拦截器栈">
        <param name="name">改名后的拦截器</param>
    </interceptor-ref>
</action>
```

聪明一点的读者应该会看出来这样写有些不妥。因为系统无法确认需要修改的 name 属性到底是该拦截器栈中的哪个拦截器，显然，这种配置方法是不成功的。实际上，通过这种方式无法修改该拦截器栈内任何拦截器的 name 属性。

对此可以采用最原始的方法：在 Action 配置中重新引用所有的拦截器，在引用第二个拦截器时修改第二个拦截器的 name 属性。代码如下。

```
<action name="Action 名" class="Action 类">
    <!-- 分别引用两个拦截器 -->
    <interceptor-ref name="拦截器一"/>
    <interceptor-ref name="拦截器二">
        <param name="name">改名后的拦截器</param>
    </interceptor-ref>
</action>
```

通过上述方式，确实可以实现修改拦截器栈内特定拦截器的属性值。但这种配置方式的缺点很明显，即如果该拦截器栈内包含了 10 个拦截器，则必须在该 Action 内重新引用 10 个拦截器——这一点，与我们一直强调的软件复用是矛盾的。

为了解决这个问题，Struts 2 提供了一种简单的修改方式：当引用拦截器栈时，直接修改该栈内特定的拦截器的属性值。代码如下。

```
<action name="Action 名" class="Action 类">
    <!-- 引用上面的拦截器栈 -->
    <interceptor-ref name="拦截器栈">
        <param name="拦截器二.name">改名后的拦截器</param>
    </interceptor-ref>
</action>
```

从上面的配置片段中可以看出，如果需要在引用拦截器栈时直接覆盖栈内某个拦截器的属性值，则在制定需要被赋给的属性时，不能只制定该属性的属性名，必须加上该属性属于的拦截器名。即采用如下方式。

```
<拦截器名>.<参数名>
```

5.3.2 实例描述

最近，做了一个企业的后台管理系统，而这个后台管理系统需要有权限控制，即很多功能是只有超级管理员才有操作权限，普通用户只有查看的权限，而这时，使用 Struts 2 中的拦截器过滤方法可以实现此功能，同时为了便于其他员工的合作将拦截结果返回，下面来为后台管理系统加一道安全锁链吧！

5.3.3 实例应用

【例 5-3】 为后台管理系统加上安全锁链。

讲这个例子之前，我先表个态吧……为了使读者清晰、客观地了解到怎样为后台管理系统加上权限控制，我在这里只是模拟一下，并没有与数据库连接，当然这也是为了方便考虑但是我的思路非常清晰，读者看完后会明白的。

(1) 既然是 Struts 2，并且是页面，Web 层是必须的。首先创建 UserAction 类，继承

ActionSupport 类，它里面有两个方法，一个是显示所有用户信息的 execute ()，另一个是删除用户信息的方法 deleteUser ()，代码如下。

```
package com.struts2.actions;
import com.opensymphony.xwork2.ActionSupport;
public class UserAction extends ActionSupport {
    //删除用户
    public String deleteUser(){
        //记录执行否
        System.out.println("我执行了 deleteUser，删除了一条用户信息，呵呵…");
        return "user";
    }
    @Override
    public String execute() throws Exception {
        //记录执行否
        System.out.println("我执行了 execute 方法");
        return "user";
    }
}
```

(2) 创建一个拦截结果的监听器类 UserPreResultListener.java，输出结果，代码如下。

```
package com.struts2.interceptor;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.PreResultListener;
public class UserPreResultListener implements PreResultListener {
    //定义在处理 Result 之前的行为
    public void beforeResult(ActionInvocation invocation, String result) {
        //打印出执行的结果
        System.out.println("返回的逻辑视图为"+result);
    }
}
```

(3) 最关键的一步，创建方法过滤的拦截器类 MyFilterInterceptor.java，记录执行过程，并把上面的监听器通过代码手动注册给拦截器，代码如下。

```
package com.struts2.interceptor;
import java.util.Date;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.MethodFilterInterceptor;
import com.struts2.actions.UserAction;
public class MyFilterInterceptor extends MethodFilterInterceptor {
    //定义拦截器的名字
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
```

```
    @Override
    protected String doIntercept(ActionInvocation invocation) throws Exception
{
        System.out.println("--------------MyFilterInterceptor 拦截器开始
--------------");
        //获取被拦截的 Action 实例
        UserAction action = (UserAction)invocation.getAction();
        System.out.println(name+"拦截器的动作:开始执行删除用户 deleteUser 的时间为:
"+new Date());
        //获取开始执行 Action 方法的开始时间
        long start = System.currentTimeMillis();
        //将一个拦截结果的监听器注册给该拦截器
        invocation.addPreResultListener(new UserPreResultListener());
        System.out.println("deleteUser 方法执行之前的拦截");
        String result = invocation.invoke();
        System.out.println(name+"拦截器的动作：执行完删除用户 deleteUser 的时间为:
" + new Date());
        //获取结束时间
        long end = System.currentTimeMillis();
        System.out.println("拦截器的动作：执行 action 事件的时间是:"+(end-start)+"
毫秒");
        System.out.println("------------结束---------------");
        return result;
    }
}
```

(4) 工作即将完成，现在将过滤方法的拦截器在 struts-config 文件夹下的 user.xml 文件中配置一下。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name ="user" extends ="struts-default" >
        <interceptors>
            <interceptor name="myfilter"
class="com.struts2.interceptor.MyFilterInterceptor">
                    <!-- 为拦截器指定参数值 -->
                    <param name="name">方法过滤拦截器</param>
            </interceptor>
        </interceptors>
        <action name ="user" class ="com.struts2.actions.UserAction" >
            <result name="user">/user.jsp</result>
            <!-- 在 Action 中引用拦截器 -->
            <interceptor-ref name="myfilter">
                <param name="name">改名后的拦截器</param>
                <!-- 指定 execute 方法不需要被拦截 -->
                <param name="excludeMethods">execute</param>
```

```
            </interceptor-ref>
        </action >
    </package >
</struts>
```

(5) 在 struts.xml 文件中引入 user.xml 文件。

(6) 编辑 user.jsp，用于显示所有的管理员列表信息。一般情况下可以从数据库中读取，得到一个用户集合，然后把这个用户集合 List 保存至 request 对象中，在页面中用 Struts 2 标签中的<s:iterator…/>遍历集合显示，这里不再写明代码。

5.3.4　运行结果

单击左边导航中的“管理员列表”，链接地址为 user/user.action，右边显示 user.jsp 页面，如图 5-6 所示。



图 5-6　管理员列表

单击图 5-6 中的“删除”按钮，执行方法过滤的拦截器类 MyFilterInterceptor.java 中的 doIntercept(ActionInvocation invocation)方法，后台输出结果如图 5-7 所示。

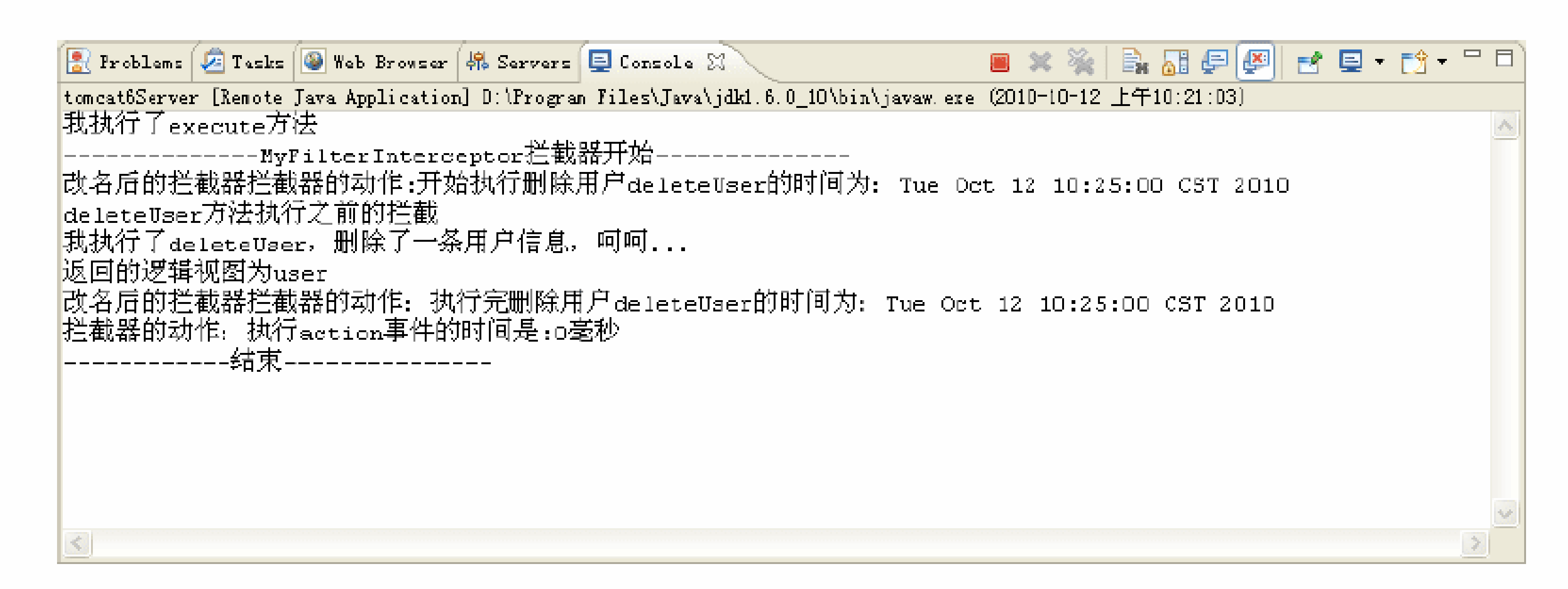


图 5-7　后台输出

5.3.5 实例分析



源码解析：

上述案例中，在 struts.xml 中配置了方法过滤拦截器类 MyFilterInterceptor.java，它要拦截的 Action 为 UserAction，而在 user.xml 文件中编辑<param name="excludeMethods"> execute</param>，使 UserAction 中的 execute()方法不要拦截，也就说只拦截 deleteUser()方法。故当执行 user/user.action 时，后台只输出“我执行了 execute()方法”，而不执行 MyFilterInterceptor 类中的 doIntercept(ActionInvocation invocation)方法，当单击“删除”按钮，执行 user/user!deleteUser.action 时，执行 MyFilterInterceptor 类中的 doIntercept(ActionInvocation invocation)方法，同时返回拦截结果“返回的逻辑视图为 user”。

5.4 Struts 2 内置拦截器

俗话说：金无足赤，人无完人。

如今，哪个软件在广大人群中已占据了不可或缺的地位？腾讯 QQ。

腾讯 QQ 空间装扮中用户可以用腾讯内部自带的色调、模块、布局……装扮自己的空间，也可以自定义上传自己想要的图片进行装扮；腾讯 QQ 头像可以用腾讯内部自带的 QQ 头像更换自己的 QQ 头像，也可以自己上传头像进行更换头像；腾讯 QQ 聊天窗体皮肤可以用腾讯内部自带的皮肤，也可以自定义自己想要的皮肤颜色……总而言之，腾讯在这一方面做到了完美。

Struts 2 作为 Web 开发中最流行的框架，它的成功和腾讯一样，是属于“实力派”的。Struts 2 中的拦截器也存在自定义拦截器和内置拦截器之分，下面将为读者讲解一下 Struts 2 的内置拦截器。



视频教学：光盘/videos/05/SystemInterceptor.avi　　　　长度：7 分钟

5.4.1 基础知识——内置拦截器

从 Struts 2 框架来看，拦截器几乎完成了 Struts 2 框架 70%的工作，包括解析请求参数，将请求参数赋值给 Action 属性，执行数据校验，文件上传…… Struts 2 设计的灵巧性，更大程度地得益于拦截器设计，当需要扩展 Struts 2 功能时，只需要提供对应的拦截器，并将它配置在 Struts 2 容器中即可；如果不需要该功能时，也只需要取消该拦截器的配置即可。这种可插拔式的设计，正是软件设计领域一直孜孜以求的目标。

Struts 2 框架的大部分工作都是通过内建拦截器完成的。

1. 内置拦截器

Struts 2 内置了大量的拦截器，这些拦截器以 key-value 对的形式配置在 struts-default.xml

文件中，其中 name 是拦截器的名字，就是后面使用该拦截器的引用点；value 则指定了该拦截器的实现类，如果定义的 package 继承了 Struts 2 的默认 struts-default 包，则可以自由使用下面的拦截器，否则必须自定义这些拦截器。

下面就来给读者介绍一下 Struts 2 的内置拦截器，如表 5-1 所示。

表 5-1　Struts 2 的内置拦截器

拦截器名	功　能
alias	实现在不同请求中相似参数别名的转换
autowiring	这是个自动装配的拦截器，主要用于当 Struts 2 和 Spring 整合时，Struts 2 可以使用自动装配的方式来访问 Spring 容器中的 Bean
chain	构建一个 Action 链，使当前 Action 可以访问前一个 Action 的属性，一般和<result type="chain" .../>一起使用
conversionError	这是一个负责处理类型转换错误的拦截器，它负责将类型转换错误从 Action Context 中取出，并转换成 Action 的 FieldError 错误
createSession	该拦截器负责创建一个 HttpSession 对象，主要用于那些需要有 HttpSession 对象才能正常工作的拦截器中
debugging	当使用 Struts 2 的开发模式时，这个拦截器会提供更多的调试信息
execAndWait	后台执行 Action，负责将等待画面发送给用户
exception	这个拦截器负责处理异常，它将异常映射为结果
fileUpload	这个拦截器主要用于文件上传，它负责解析表单中文件域的内容
i18n	这是支持国际化的拦截器，它负责把所选的语言、区域放入用户的 Session 中
logger	这是一个负责日志记录的拦截器，主要是输出 Action 的名字
model-driven	这是一个用于模型驱动的拦截器，当某个 Action 类实现了 ModelDriven 接口时，它负责把 getModel()方法的结果放入 ValueStack 中
scoped-model-driven	如果一个 Action 实现了一个 ScopedModelDriven 接口，该拦截器负责从指定生存范围中找出指定的 Model，并通过 setModel()方法将该 Model 传给 Action 实例
params	这是一个最基本的拦截器，它负责解析 HTTP 请求中的参数，并将参数值设置成 Action 对应的属性值
prepare	如果 Action 实现了 Preparable 接口，将会调用该拦截器的 prepare()方法
static-params	这个拦截器负责将 xml 中<action>标签下的<param>标签中的参数传入 Action
scope	这是范围转换拦截器，它可以将 Action 状态信息保存到 HttpSession 范围，或者保存到 ServletContext 范围中
servlet-config	如果某个 Action 需要直接访问 Servlet API，就是通过这个拦截器实现的
roles	这是一个 JAAS(Java Authentication and Authorization Service，Java 授权和认证服务)拦截器，只有当浏览者取得合适的授权后，才可以调用被该拦截器拦截的 Action
timer	这个拦截器负责输出 Action 的执行时间，这个拦截器在分析该 Action 的性能瓶颈时比较多用

续表

拦截器名	功　能
token	这个拦截器主要用于阻止重复提交，它检查到 Action 中的 token，从而防止多次提交
token-session	这个拦截器的作用与前一个基本类似，只是它把 token 保存在 HttpSession 中
validation	通过执行在 xxxAction-validation.xml 中定义的校验器，从而完成数据校验
workflow	这个拦截器负责调用 Action 类中的 validate()方法，如果校验失败，则返回 input 的逻辑视图



尽量避免在 Action 中直接访问 Servlet API，否则会导致 Action 与 Servlet 的高耦合。

大部分的时候，开发者无需手动控制这些拦截器，因为 struts-default.xml 文件中已经配置了这些拦截器，只要定义的包继承了系统的 struts-default 包，就可以直接使用这些拦截器。

2．struts-default.xml文件中的拦截器配置

struts-default.xml 文件是 Struts 2 默认的配置文件，不管在什么时候，这个配置文件都会自动加载。Struts 2 大部分的内置拦截器都配置在该文件中(还有少量的内置拦截器配置在 Struts 2 的插件配置文件中)。

```
    <interceptor name="alias"
class="com.opensymphony.xwork2.interceptor.AliasInterceptor"/>
    <interceptor name="autowiring"
class="com.opensymphony.xwork2.spring.interceptor.
ActionAutowiringInterceptor"/>
    <interceptor name="chain"
class="com.opensymphony.xwork2.interceptor.ChainingInterceptor"/>
    <interceptor name="conversionError"
class="org.apache.Struts2.interceptor.StrutsConversionErrorInterceptor"/>
    <interceptor name="cookie"
class="org.apache.struts2.interceptor.CookieInterceptor"/>
    <interceptor name="clearSession"
class="org.apache.struts2.interceptor.ClearSessionInterceptor" />
    <interceptor name="createSession"
class="org.apache.struts2.interceptor.CreateSessionInterceptor" />
    <interceptor name="debugging"
class="org.apache.struts2.interceptor.debugging.DebuggingInterceptor" />
    /*省略更多的拦截器配置*/
    <interceptor name="validation"
class="org.apache.struts2.interceptor.validation.
AnnotationValidationInterceptor"/>
    <interceptor name="workflow"
class="com.opensymphony.xwork2.interceptor.DefaultWorkflowInterceptor"/>
    <interceptor name="store"
class="org.apache.struts2.interceptor.MessageStoreInterceptor" />
    <interceptor name="checkbox"
class="org.apache.struts2.interceptor.CheckboxInterceptor" />
```

```
    <interceptor name="profiling"
class="org.apache.struts2.interceptor.ProfilingActivationInterceptor" />
    <interceptor name="roles"
class="org.apache.struts2.interceptor.RolesInterceptor" />
    <interceptor name="jsonValidation"
class="org.apache.struts2.interceptor.validation.JSONValidationInterceptor"
/>
    <interceptor name="annotationWorkflow"
class="com.opensymphony.xwork2.interceptor.annotations.
AnnotationWorkflowInterceptor" />
    <interceptor name="multiselect"
class="org.apache.struts2.interceptor.MultiselectInterceptor" />
```

从上面的配置片段中可以看出，Struts 2 内置的拦截器同样采用了前面介绍的<interceptor .../>元素来配置。当配置了这一系列原始的拦截器以后，Struts 2 可利用这些拦截器组合一系列的拦截器栈。配置拦截器栈的代码如下。

```
<!—最基本功能拦截器栈-->
<interceptor-stack name="basicStack">
    <interceptor-ref name="exception"/>
    <interceptor-ref name="servletConfig"/>
    <interceptor-ref name="prepare"/>
    <interceptor-ref name="checkbox"/>
    <interceptor-ref name="multiselect"/>
    <interceptor-ref name="actionMappingParams"/>
    <interceptor-ref name="params">
        <param name="excludeParams">dojo\..*,^struts\..*</param>
    </interceptor-ref>
    <interceptor-ref name="conversionError"/>
</interceptor-stack>
<!—在最基本功能的基础上增加了数据校验拦截器的拦截器栈-->
<interceptor-stack name="validationWorkflowStack">
    <interceptor-ref name="basicStack"/>
    <interceptor-ref name="validation"/>
    <interceptor-ref name="workflow"/>
</interceptor-stack>
<!—基本的 JSON 验证拦截器栈 -->
<interceptor-stack name="jsonValidationWorkflowStack">
    <interceptor-ref name="basicStack"/>
    <interceptor-ref name="validation">
        <param name="excludeMethods">input,back,cancel</param>
    </interceptor-ref>
    <interceptor-ref name="jsonValidation"/>
    <interceptor-ref name="workflow"/>
</interceptor-stack>
/*省略其他的拦截器栈配置*/
<!—这是系统默认的拦截器栈，该拦截器栈可以满足大部分的 Struts 2 应用的需要-->
<interceptor-stack name="defaultStack">
    <interceptor-ref name="exception"/>
```

```
    <interceptor-ref name="alias"/>
    <interceptor-ref name="servletConfig"/>
    <interceptor-ref name="i18n"/>
    <interceptor-ref name="prepare"/>
    <interceptor-ref name="chain"/>
    <interceptor-ref name="debugging"/>
    /*省略其他的拦截器引用*/
    <interceptor-ref name="validation">
        <param name="excludeMethods">input,back,cancel,browse</param>
    </interceptor-ref>
    <interceptor-ref name="workflow">
        <param name="excludeMethods">input,back,cancel,browse</param>
    </interceptor-ref>
</interceptor-stack>
<!—这是系统默认拦截器栈的一个别名，定义这个拦截器栈仅仅是为了保持向后兼容-->
<interceptor-stack name="completeStack">
    <interceptor-ref name="defaultStack"/>
</interceptor-stack>
```

上面的配置片段配置了开发 Struts 2 应用所需要的大部分拦截器栈，大部分时候，直接使用系统的 defaultStack 拦截器栈即可，虽然它可能会做一些额外的拦截工作，但是对系统不会有太大的影响。当然如果读者有非常大的把握，也可以为每个 Action 配置分别指定拦截器，但这种工作量会非常大。



通常不推荐为每个 Action 分别定义拦截器，而是推荐直接使用系统的 defaultStack 拦截器栈。

用户无须配置自己的拦截器，甚至无须配置任何拦截器引用，因为 Struts 2 将 defaultStack 拦截器栈配置成默认的拦截器栈。

```
<!-- 将 defaultStack 拦截器栈配置成默认的拦截器栈 -->
<default-interceptor-ref name="defaultStack"/>
```

因为 Struts 2 的 struts-default 包中指定 defaultStack 拦截器栈时默认的拦截器栈，因此如果用户定义的包继承了 struts-default 包，也会将 defaultStack 拦截器栈作为默认的拦截器栈。这意味着：如果系统中的 Action 配置没有指定拦截器引用，系统会将 defaultStack 拦截器栈自动作用于该 Action。



也许是因为开发者头脑比较灵活吧，一般开发者都喜欢用自己定义的拦截器，因此自定义拦截器在开发中很是“吃香”啊！不过，读者如果用 Struts 2 的内置拦截器能满足需求，就不要使用自定义拦截器了，因为这样可以节省时间。

5.4.2 实例描述

前几天看了一个太极拳的网站，其中讲述了很多太极拳的拳法，我很是受益。当然可能

是“职业病”的缘故吧，我看网站重点并不在于网站的文章内容，而是在于网站的设计，其中有一个功能就是每篇文章都有一个“访问量”统计，乍一看：哇噻……个、十、百、千、万……这么多人在浏览。看来能接受这么大的访问量。它的服务器质量还不错哦！在我无意按下 F5 刷新键之下，再次回头看文章访问量，次数加了一。心想可能是又有一个人访问吧，出于优秀开发者的角度，我还是试了一下，再次按下 F5，没想到，访问量又加一，我按 F5，它加一。

我按、按、按，它加、加、加……怪不得那么多的访问量，原来如此啊……

5.4.3　实例应用

【例 5-4】 刷新页面，不提交表单。

(1)　新建用户实体类 User.java，内容如下。

```
package com.struts2.model;
public class User {
    private String username;//用户名
    private String realname;//真实姓名
    private String pass;//密码
    private String phone;//联系电话
    /*下面是上面所有属性的 get、set 方法，这里省略*/
    …
}
```

(2)　修改 UserAction 类，记录新添加的管理员信息，并保存至 List<User>中，同时把所有添加的管理员信息保存至 application 中，以供页面显示，修改后的代码如下。

```
package com.struts2.actions;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
import com.struts2.model.User;
public class UserAction extends ActionSupport {
    //定义用户集合
    List<User> userList=new ArrayList<User>();
    private User user;
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
    //删除用户
    public String deleteUser(){
```

```
        System.out.println("我执行了 deleteUser，删除了一条用户信息，呵呵…");
        return "user";
    }
    @Override
    public String execute() throws Exception {
        System.out.println("我执行了 execute 方法");
        return "user";
    }
    //打开添加用户信息
    public String addUser(){
        return "add_user";
    }
    //添加用户信息
    public String add(){
        //获取 ActionContext 对象
        ActionContext context = ActionContext.getContext();
        //获取 application
        Map application = context.getApplication();
        //获取 session
        Map session = context.getSession();
        //把刚添加的用户信息保存至集合中
        userList.add(user);
        //判断是否是第一次添加用户，如果不是遍历 application 中的集合，并添加至用户集合中
    if(application.get("users")!=null&&((List<User>)application.get("users"
)).size()>0){
            for(int
i=0;i<((List<User>)application.get("users")).size();i++){

    userList.add(((List<User>)application.get("users")).get(i));
            }
        }
        //把用户集合放入 application 中
        application.put("users", userList);
        session.put("userlist", userList);
        return "add_user";
    }
}
```

(3) 修改 src 目录下的 struts-config 文件夹中的 user.xml 文件，把原来的方法过滤拦截器删除，使用 Struts 2 的内置拦截器 token，并配置拦截方法为 UserAction 中的 add()方法，修改后的 user.xml 文件内容如下所示。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
```

```
    <package name ="user" extends ="struts-default" >
        <action name ="user" class ="com.struts2.actions.UserAction" >
            <result name="user">/user.jsp</result>
            <result name="add_user">/add_input.jsp</result>
            <!—配置刷新页面执行 add 方法时跳转页面-->
            <result name="invalid.token">/tokenError.jsp</result>
            <!-- 使用默认的拦截器 -->
            <interceptor-ref name="defaultStack"/>
            <!-- 配置 Struts2 内置拦截器 -->
            <interceptor-ref name="token">
                <!-- 配置只拦截的方法为 add 方法 -->
                <param name="includeMethods">add</param>
            </interceptor-ref>
        </action >
    </package >
</struts>
```

(4) 在 user.jsp 页面中添加按钮，单击跳转至添加管理员信息页面 add_input.jsp。

```
<a href="user/user!addUser.action">添加管理员</a>
```

(5) 编辑 add_input.jsp 页面，使用 Struts 2 标签中<s:token/>标签，防止页面刷新提交表单，代码如下。

```
<%@taglib prefix="s" uri="/struts-tags"%>
        <s:form action="user/user!add.action" method="post">
                <s:textfield name="user.username" label="用户名">
                </s:textfield>
                <s:textfield name="user.realname" label="真实姓名">
                </s:textfield>
                <s:textfield name="user.pass" label="密码"></s:textfield>
                <s:textfield name="user.phone" label="联系电话"></s:textfield>
                <s:token/>
                <s:submit value="  提交    "></s:submit>
        </s:form>
```

(6) 细心的读者在上面的 user.xml 文件配置中，已经看到了这个功能还需要一个页面 tokenError.jsp，即用于当用户刷新页面，并且要执行拦截器 token 所拦截的方法 add()时要跳转的页面。

5.4.4　运行结果

单击 user.jsp 页面上的“添加管理员”按钮，跳转至添加管理员输入信息页面 add_input.jsp，如图 5-8 所示。



图 5-8　添加管理员输入信息页面

单击“提交”按钮，查看管理员列表页面，如图 5-9 所示，添加了一条新记录。



图 5-9　添加一条新记录

再次单击“添加管理员”按钮，并输入管理员信息，先单击“提交”按钮，这时管理员列表肯定已经存在你刚输入的管理员信息了。刷新页面，会出现一个提示框，选择“重试”，出现如图 5-10 所示的画面，即 tokenError.jsp 页面。



图 5-10　重复提交表单跳转至tokenError.jsp页面

看看管理员列表信息，是否有刷新过后的提交数据呢？没有，这就是 token 拦截器起的作用。

5.4.5　实例分析



源码解析：

在上述案例中，我们在 user.xml 中配置 token 拦截器时，设置了 token 拦截器只拦截 UserAction 的 add()方法，使用了 includeMethods 参数，因此当单击左边导航“管理员列表”和单击 user.jsp 页面中的任何一个按钮时都不会被 token 拦截器拦截，而当我们要添加管理员信息时，也就是执行 add()方法时会被 token 拦截器拦截，由于 token 拦截器一般与 Struts 2 标签 <s:token/>同时存在，所以共同完成了刷新页面不提交表单的功能，这就是 Struts 2 内置拦截器 token 所起的作用。

5.5　使用拦截器完成权限控制

前面在讲自定义拦截器的时候，说到了“门”，并且也举了例子：为程序做扇“门”。但是前面的着重点在于讲解自定义拦截器，这次将用一节的内容来讲解如何为程序做扇“门”？如何判断用户是否有足够的权限来执行程序中的某些操作？这也是考虑到在实际开发中经常要用的缘故，希望读者能仔细阅读这一节内容，相信你会发现很多奥妙的！



视频教学： 光盘/videos/05/TextInterceptor.avi　　　　长度：8 分钟

5.5.1　基础知识——实现权限控制拦截器

大部分 Web 应用都涉及权限控制，当浏览者需要请求执行某个操作时，应用需要先检查浏览者是否登录，以及是否有足够的权限来执行该操作。

本实例应用要求用户登录，且必须为指定用户名才可以查看系统中某个视图资源；否则，系统直接转入登录页面。对于上面的请求，可以在每个 Action 的执行实际处理逻辑之前，先执行权限检查逻辑，但这种做法不利于代码复用。因为大部分 Action 里的权限检查代码都大同小异，故将这些权限检查的逻辑放在拦截器中进行检查将会更加优雅动人。

检查用户是否登录，通常都是通过跟踪用户的 Session 来实现，通过 ActionContext 即可访问到 Session 中的属性。

```
//获取请求相关的 ActionContext 实例
ActionContext context=invocation.getInvocationContext();
//获取 session
Map session = context.getSession();
```

读者有没有觉得很眼熟啊？前面我们曾经获取过 Session，和这里的略有不同。拦截器的 intercept(ActionInvocation invocation)方法中的 invocation 参数可以很轻易地访问到请求相关的 ActionContext 实例。

这里要强调一点，也是前面从来没有提到过的。在拦截器的 intercept(ActionInvocation invocation)方法中，可以在用户没有登录时直接返回 login 的逻辑视图，代码如下所示。

```
return Action.LOGIN;
```

一旦实现了一个完美的权限检查拦截器，就可以在所有需要实现权限控制的 Action 中复用已经定义好的拦截器。

考虑到这个拦截器的重复使用，可能多个 Action 都需要跳转到 login 逻辑视图，故将 login Result 定义成一个全局 Result。下面是配置 login Result 的配置片段。

```
<!-- 定义全局 Result -->
<global-results>
        <!-- 当返回 login 视图名时，转入/login.jsp 页面 -->
        <result name="login">/login.jsp</result>
</global-results>
```



经过上面的配置，当用户没有登录直接访问系统中的页面时，将转入 login.jsp 页面。

这种通过拦截器进行权限控制的方式，显然具有更好的代码复用性。

如果为了简化 struts.xml 文件的配置，避免在每个 Action 中重复配置该拦截器，可以将该拦截器配置成一个默认拦截器栈(这个默认拦截器栈应该包括 default-stack 拦截器栈和权限检查拦截器)。

定义自己默认的拦截器栈的配置如下。

```
<interceptors>
    <!-- 定义权限检查拦截器 -->
    <interceptor name="权限检查拦截器名" class="权限检查拦截器类"/>
    <!-- 定义一个包含权限检查的拦截器栈 -->
    <interceptor-stack name="自己默认的拦截器栈名 mydefault">
        <!-- 定义拦截器栈包含 default-stack 拦截器栈 -->
        <interceptor-ref name="default-stack"/>
        <!-- 定义拦截器栈包含权限检查拦截器 -->
        <interceptor-ref name="权限检查拦截器名"/>
    </interceptor-stack>
</interceptors>
```

一旦定义了上面的 mydefault 拦截器栈，这个拦截器栈就包含了权限检查拦截器和系统默认的拦截器栈。如果将这个拦截器栈定义成默认拦截器，则可以避免在每个 Action 中重复定义权限检查拦截器。下面是定义默认拦截器的配置片段。

```
<default-interceptor-ref name="mydefault"/>
```

一旦在某个包下定义了上面的默认拦截器栈，在该包下的所有 Action 都会自动增加权限检查功能。对于那些不需要使用权限控制的 Action，将它们定义在另外的包中——这个包中依然使用系统原来的默认拦截器栈，将不会有权限控制功能。

5.5.2　实例描述

前几天听朋友说了一个可恶至极的事情，现在想起来都觉得那人太可怕了。

事情是这样的，他们公司有一个员工性格比较内向，不太爱说话，他们老板更是看见他就烦，几乎是五天一大吵，三天一小吵。本来我听到这还挺同情这人的，内向又不是错，现在这社会内向虽然不吃香，不如能说会道的好，但是内向不是人家的错，只要人家工作踏实、勤奋就好……，可是下面可恨的事情来了，结果老板在人家到公司后的第五个月把他给开除了，也许，那个内向的员工一肚子的火没地方撒吧！在他收拾东西的过程中，他竟然登录了公司内部的管理系统，把东西给删了个精光。我的妈啊！我真不知该说是这个公司老板的错，还是这个内向员工的错，你们评评理，这是谁的错……

5.5.3　实例应用

【例 5-5】 让非超级管理员无权限。

(1) 修改 LoginAction.java 类，判断用户输入的用户名是否是 admin，如果是把用户信息保存至 Session 中，否则返回登录页面。修改后的内容如下。

```
package com.struts2.actions;
import java.util.Map;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
import com.struts2.model.User;
public class LoginAction extends ActionSupport {
    private static final long serialVersionUID = -4278369739741451645L;
    private User user;
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
@Override
    public String execute() throws Exception {
        //获取相关的 ActionContext
        ActionContext context=ActionContext.getContext();
        //获取 Session
        Map session=context.getSession();
        //判断用户输入的用户名是否是 admin，如果是，存入 session 中，否则返回登录页面，
          重新输入
```

```
        if(user!=null&&user.getUsername().equals("admin")){
            session.put("user", user);
        }
        else {
            return "input";
        }
        return "main";
    }
}
```

(2) 去掉 struts-config 文件夹下 login.xml 文件中配置的 LoginAction 类的拦截器 SimpleInterceptor，修改为：

```
<action name ="login" class ="com.struts2.actions.LoginAction" >
    <result name="main">/main.jsp</result>
    <result name="input">/login.jsp</result>
</action>
```

登录时，并不用拦截器，只需要在 LoginAction 类中对用户输入信息作个简单的验证即可。

(3) 修改拦截 UserAction 的拦截器类 SimpleInterceptor.java，判断用户输入的是否是超级管理员信息，如果不是返回无权限页面。修改后的 SimpleInterceptor.java 为：

```
package com.struts2.interceptor;
import java.util.Map;
import com.opensymphony.xwork2.Action;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.AbstractInterceptor;
import com.struts2.model.User;
public class SimpleInterceptor extends AbstractInterceptor {
    //定义拦截器名称
    private String name;
    public void setName(String name)
    {
        this.name = name;
    }
    public String intercept(ActionInvocation invocation)throws Exception
    {
        //取得请求相关的 ActionContext 实例
        ActionContext context=invocation.getInvocationContext();
        Map session=context.getSession();
        //取出名为 user 的 Session 属性
        User user=(User)session.get("user");
        //如果没有登录，或者登录所用的用户名或者密码不是 admin，都返回重新登录
    if(user!=null&&user.getUsername().equals("admin")&&user.getPass().
equals("admin"))
        {
```

```
            return invocation.invoke();
        }
        //没有登录，将服务器提示设置成一个 HttpServletRequest 属性
        //直接返回 error 的逻辑视图
        return Action.ERROR;
    }
}
```

(4) 修改 struts-config 文件夹下的 user.xml 文件，为 UserAction 类配置拦截器 SimpleInterceptor.java，修改后为：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name ="user" extends ="struts-default" >
        <interceptors>
            <!-- 权限控制拦截器 -->
            <interceptor name="authority"
class="com.struts2.interceptor.SimpleInterceptor"/>
        </interceptors>
        <!-- 定义全局 Result -->
        <global-results>
            <!-- 当返回 error 视图名时，转入/error.jsp 页面 -->
            <result name="error">/error.jsp</result>
        </global-results>
        <action name ="user" class ="com.struts2.actions.UserAction" >
            <result name="user">/user.jsp</result>
            <result name="add_user">/add_input.jsp</result>
            <!-- 定义权限控制拦截器 -->
            <interceptor-ref name="defaultStack"/>
            <interceptor-ref name="authority"/>
        </action >
    </package >
</struts>
```

(5) 再把登录页面 login.jsp 页面中的用户名输入框的 name 属性值由原来的 username 改成 user.username；密码输入框的 name 属性值由 pass 改为 user.pass。

(6) 新建 error.jsp 页面，提示“无权限”即可。

5.5.4　运行结果

运行程序，在登录页面中输入用户名：admin，口令：123456789。单击“登录”按钮，进入主页面，单击左边导航中的“管理员列表”，出现提示无权限信息页面，如图 5-11 所示。



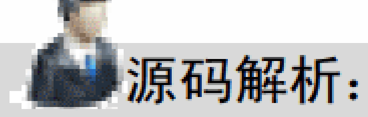
图 5-11　非超级管理员权限

返回登录页面，输入用户名：admin，口令：admin。登录之后再次单击左边导航中的“管理员列表”显示所有管理员信息，如图 5-12 所示。



图 5-12　超级管理员权限

5.5.5　实例分析



源码解析：

上述案例中，在 LoginAction 类中并未定义拦截器，而是在它的 execute()方法中对用户输入信息进行了一下简单验证：只要用户名为 admin 的就可以登录本系统。但是登录到本系统的未必能对本系统的所有操作都有权限，因此需要为 UserAction 类定义一个拦截器 SimpleInterceptor，验证只有是超级管理员才有查看管理员列表的权限。

这个功能在实际开发中应用很多，不同的用户身份登录系统后权限是不一样的。

5.6　使用拦截器注解

如果您是一个开发者的话，一定深知“注解”为您带来的惊喜吧！在 Hibernate 中，可以利用注解来进行映射；在 Spring 中，可以利用注解来实现 AOP；今天，同样可以利用拦截器注解来指定在 Struts 2 中的 Action 执行之前和之后需要调用的方法。

下面这一节就来为读者讲解一下如何使用拦截器注解。


视频教学： 光盘/videos/05/annotations.avi　　　　长度：7 分钟

5.6.1　基础知识——使用拦截器注解

Struts 2 在 com.opensymphony.xwork2.interceptor.annotations 包中定义了 3 个拦截器注解类型，读者可以不用编写拦截器类，直接通过注解的方式来指定在 Action 执行之前和之后需要调用的方法。

Struts 2 提供的 3 个拦截器注解类型都只能应用到方法级别，如下所示。

1．Before

Before 用于标注一个 Action 方法，该方法将在 Action 的主要方法(如 execute()方法)调用之前调用。如果标注的方法有返回值，并且不为 null，那么它的返回值将作为 Action 的结果代码。

2．After

After 用于标注一个 Action 方法，该方法将会在 Action 的主要方法以及 result 执行之后调用。如果标注的方法有返回值，那么这个返回值将会被忽略。

3．BeforeResult

BeforeResult 用于标注一个 Action 方法，该方法将在 Action 的主要方法调用之后，在 result 执行之前调用。如果标注的方法有返回值，那么这个返回值将被忽略。

这 3 个注解类型都有一个同名的参数，该参数的作用也是相同的，如表 5-2 所示。

表 5-2　Before、After和BeforeResult注解的同名参数

参　数	类　型	是否必须	默 认 值	描　述
priority	int	否	10	指定方法执行的优先级顺序

同一个注解可以用来标注多个方法，方法执行的先后顺序可以通过 priority 参数来指定，优先级越高，方法越优先执行。在相同优先级的情况下，方法执行的顺序将无法保证。不过，对于有继承关系的 Action 类，在基类上标注的方法将优先于在派生类上标注的方法执行。



Before 和 After 注解用于替代拦截器实现，BeforeResult 注解用于替代 PreResult Listener 实现。

要使用拦截器注解，还需要配置 AnnotationWorkflowInterceptor 拦截器。这个拦截器在 struts-default.xml 文件中并未定义，因此需要用户自己在 struts.xml 文件中定义，代码如下。

```
    <interceptors>
        <!-- 配置 AnnotationWorkflowInterceptor 拦截器 -->
        <interceptor name="annotationInterceptor"
class="com.opensymphony.xwork2.interceptor.annotations.
AnnotationWorkflowInterceptor"/>
        <interceptor-stack name="annotatedStack">
            <interceptor-ref name="defaultStack"/>
            <interceptor-ref name="annotationInterceptor"/>
        </interceptor-stack>
    </interceptors>
```

采用上面的配置后，在 Action 中就可以直接引用 annotatedStack 拦截器栈了。

5.6.2 实例描述

上次公司技术部在开会的时候，提到拦截器的使用方式，一部分人说：采用自定义拦截器好，在 Struts 2 配置文件中配置一下就可以了，很方便；另一部分人说：采用拦截器注解方式好，连编辑自定义拦截器类都不用，在 Action 中的方法中编辑验证即可，查看代码那是方便又快捷啊！只要打开 Action 就知道这个 Action 想要验证哪些信息，而不像配置拦截器那样，还得先看看配置文件才知道这个 Action 用了哪个拦截器。

那个说使用拦截器注解的同事很快就给我们做了一个例子，让我们心服口服啊！

5.6.3 实例应用

【例 5-6】 使用拦截器注解进一步完善那道“门”。

(1) 修改 LoginAction.java 类，使用拦截器注解形式来验证用户信息，如果用户输入正确则保存至 Session 中，否则重新登录。修改代码如下所示。

```
package com.struts2.actions;
import java.util.Map;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.interceptor.annotations.Before;
import com.struts2.model.User;
public class LoginAction extends ActionSupport {
    private static final long serialVersionUID = -4278369739741451645L;
    private User user;
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
```

```
    }
    @Before
    public String checkUser(){
        //获取相关的 ActionContext
        ActionContext context=ActionContext.getContext();
        //获取 Session
        Map session=context.getSession();
        //判断用户输入的用户名是否是 admin，如果是，存入 session 中，否则返回登录页面，
重新输入
    if(user!=null&&user.getUsername().equals("admin")&&user.getPass().equals
("admin")){
            session.put("user", user);
            return null;
        }
        else {
            return INPUT;
        }
    }
    @Override
    public String execute() throws Exception {
        //获取相关的 ActionContext
        ActionContext context=ActionContext.getContext();
        //获取 Session
        Map session=context.getSession();
        //检查用户是否登录
        User loginUser=(User)session.get("user");
        if(loginUser!=null){
            return "main";
        }
        else {
            return INPUT;
        }
    }
}
```

(2)　在 login.xml 文件中配置 AnnotationWorkflowInterceptor 拦截器，并在 LoginAction 中引用该拦截器，配置代码如下。

```
    <package name ="login" extends ="struts-default" >
        <interceptors>
            <interceptor name="annotationInterceptor"
class="com.opensymphony.xwork2.interceptor.annotations.
AnnotationWorkflowInterceptor"/>
            <interceptor-stack name="annotatedStack">
                <interceptor-ref name="defaultStack"/>
                <interceptor-ref name="annotationInterceptor"/>
            </interceptor-stack>
        </interceptors>
        <action name ="login" class ="com.struts2.actions.LoginAction" >
            <result name="main">/main.jsp</result>
            <result name="input">/login.jsp</result>
```

```
            <!—引用 AnnotationWorkflowInterceptor 拦截器配置-->
            <interceptor-ref name="annotatedStack"/>
        </action >
    </package >
```

(3) 修改 UserAction.java 类，在执行 execute()方法之前，先判断用户是否登录，如果没有登录，则返回无权限页面，这样是为了防止浏览者直接请求 Action 方法而导致的一些不必要的麻烦，修改后的内容如下所示。

```
package com.struts2.actions;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.interceptor.annotations.Before;
import com.struts2.model.User;
public class UserAction extends ActionSupport {
    List<User> userList=new ArrayList<User>();
    private User user;
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
    @Before
    public String checkUser(){
        //获取相关的 ActionContext
        ActionContext context=ActionContext.getContext();
        //获取 Session
        Map session=context.getSession();
        User loginUser=(User)session.get("user");
        if(loginUser!=null){
            return null;
        }
        else {
            return "error";
        }

    }
    @Override
    public String execute() throws Exception {
        /*内容省略*/
    }
    /*打开添加用户信息方法省略*/
    /*添加用户信息方法省略*/
    /*删除用户信息方法省略*/
}
```

(4) 接着在 UserAction 中配置 AnnotationWorkflowInterceptor 拦截器，和上面在 LoginAction 中配置一样，先在 package 的名称中为 user 定义拦截器栈 annotatedStack，然后在 UserAction 中引用就可以了，同时去掉权限控制拦截器的配置。

5.6.4　运行结果

运行 login.jsp 页面，输入用户名和口令，如果输入的用户名或口令不是 admin，则无法进入主页面 main.jsp。

直接访问 login/login.action，看看出现什么结果？对，还是返回 login.jsp 页面。

接着直接访问 user/user.action，看看出现什么结果？结果如图 5-13 所示。



图 5-13　直接访问Action

5.6.5　实例分析



源码解析：

上述案例中，使用拦截器注解形式为 Action 做了拦截，验证用户是否已经登录并且是合法用户。在执行 LoginAction 中的主方法 execute()之前会调用 checkUser()方法，所以如果直接访问 login/login.action 时，LoginUser 中的 User 对象是空的，无法把用户输入信息保存至 Session 中，故返回“INPUT”结果页面，对应的就是 login.jsp 页面。

在 UserAction 中新添加方法 checkUser()，并使用“@Before”对该方法作了注解，程序在执行 UserAction 中的 execute()方法之前会执行 checkUser()方法；如果用户未登录，Session 中的“user”是 null，故返回“error”结果页面，对应的就是 error.jsp 页面，即提示无权限页面。

5.7 常见问题解答

5.7.1 Struts 2 自带的拦截器已经很强大，是否可以不用自定义拦截器



Struts 2 自带的拦截器不是已经很强大了吗？是不是就不用自定义拦截器了？
网络课堂：http://bbs.itzcn.com/thread-11003-1-1.html

问：不是说 Struts 2 自带的拦截器已经很强大了吗？那为什么还要用自定义的拦截器呢？只用 Struts 2 自带的拦截器是不是在开发中已经够用了啊？如果想写一个检查用户是否登录的拦截器该如何下手呢？

答：首先我要回答你的是一个字，对，是需要我们自定义拦截器去实现。

编辑一个检查用户是否登录的拦截器是开发中最常用的拦截器之一，思路如下。

(1) 将用户名、密码等信息记录在 Session 中。

(2) 编写自定义拦截器，取出 Session 进行判断，看是否存在该用户信息。

(3) 将拦截器配置到 struts.xml 相应的 Action 中。

自定义拦截器的方法要继承 AbstractInterceptor，实现抽象方法。

```
public abstract String intercept(ActionInvocation invocation) throws Exception
```

具体的例子我在这章已经做了重点讲解了，仔细阅读本章内容你将会明白拦截器的具体配置和使用。

5.7.2 Struts 2 拦截器的错误信息如何显示在页面上



Struts 2 拦截器的错误信息如何显示在页面上？
网络课堂：http://bbs.itzcn.com/thread-11005-1-1.html

问：我在拦截器类中把错误信息放入一个容器中，但是我怎么在页面上显示呢？比如下面这段代码里的“ctx.put(“tip”, “您还没有登录，请重新登录”);”，我怎么能在页面上显示呢？

```
    public class AuthorityInterceptorForAdmin extends AbstractInterceptor {
    private static final long serialVersionUID = 1358600090729208361L;
    // 拦截 Action 处理的拦截方法
    public String intercept(ActionInvocation invocation) throws Exception {
        // 取得请求相关的 ActionContext 实例
        ActionContext ctx = invocation.getInvocationContext();
        Map session = ctx.getSession();
```

```
        // 取出名为 user 的 session 属性
        String user = (String) session.get("aid");
        List<Admin> adminlist = new Base().getAdmin().getAdminAll();
        Iterator it = adminlist.iterator();
        while (it.hasNext()) {
            Admin admin = (Admin) it.next();
            String ad = admin.getName();
            if (user != null && user.equals(ad)) {
                return invocation.invoke();
            }
        }
        // 没有登录，将服务器提示设置成一个 HttpServletRequest 属性
        ctx.put("tip", "您还没有登录，请登陆系统");
        return Action.LOGIN;
    }
}
```

答：首先，你的拦截器应该有一个返回页面，也就是说触发拦截时要返回到的页面(比方说登录页面)。这个页面你可以自己在配置文件里定义，用过 Struts 2 的校验功能，就应该知道如何在页面显示提示信息了。(针对 Struts 2 的重写方法校验，不是 XML 校验框架)。简单来说就是：用 Action 中的 addFieldError()方法。

JSP 页面上用下列代码即可显示。

```
<s:fielderror></s:fielderror>
```

当然，显示提示信息可以灵活运用，你完全可以把信息设置在自定义的控件上。

5.7.3 Struts 2 拦截器后跳转页面问题



Struts 2 拦截器后跳转页面问题！
网络课堂：http://bbs.itzcn.com/thread-11006-1-1.html

问：我用了自编的拦截器，如果输入错误的 Action 就返回 Action.LOGIN，在 struts.xml 中，设置 flash 不能显示。但是我把<result name="login">/index.jsp</result>改成<result name="login" type="redirect">/index</result>，结果就没事了。这里怎么回事？

答：因为跳转时服务器端跳转，地址栏没改变，所以会造成路径错误，建议你将图片等路径改为以下形式。

```
<img src="#request.path"/images/xxx.gif/>
```

其中/images/xxx.gif 为你的图片在 WebRoot 下的路径和文件名。

5.7.4 Struts 2 拦截器通俗点到底是什么？为什么要用



自学中，Struts 2 拦截器通俗点到底是什么东东啊？为什么要用？
网络课堂：http://bbs.itzcn.com/thread-11011-1-1.html

问：最近自学中，用是全会了，但是我具体还是不怎么理解。为什么要用？有什么好处？什么时候用最恰当？亦或者就拿登录页面打个比方。

答：拦截器相当于过滤器，就是把你不想要的或不想显示的内容给过滤器。拦截器可以抽象出一部分代码，用来完善原来的 Action，同时可以减轻代码冗余，提高重用率。

例如在登入一个页面时，如果要求用户密码、权限等的验证，就可以用自定义的拦截器进行密码验证和权限限制。对符合的登入者才跳转到正确页面。这样如果有新增权限的话，不用在 Action 里修改任何代码，直接在 Interceptor 里修改就行了。

5.8 习　题

一、填空题

(1) 定义拦截器最简单的格式为：

```
<interceptor name="__________" class="拦截器类"/>
```

(2) 如果一个 action 需要多个拦截器，在 action 中的配置为 <interceptor-ref name="loginAndSecurityStack"/>，则在<interceptors>元素内的配置为。

```
<interceptors>
        <interceptor name="logger"
class="com.struts2.interceptor.LoginInterceptor"/>
        <interceptor name="security"
class="com.struts2.interceptor.SecurityInterceptor"/>
        __________
</interceptors>
```

(3) 如果用户要开发自己的拦截器类，应该实现 com.opensymphony.xwork2.interceptor.Interceptor 接口，该接口中的__________方法是用户需要实现的拦截动作。

(4) 我们在实现方法过滤拦截器时需要编辑一个继承 MethodFilterInterceptor.java 的类，同时需要重写父类中的__________方法。

(5) Struts 2 提供的 3 个拦截器注解类型中，它们都有一个同名的参数，该参数的名字是__________。

二、选择题

(1) 我们在定义一个拦截器时可以为该拦截器指定参数及参数值，用<param></param>标签来指定，那么我们最多可以为一个拦截器指定多少个参数呢？__________

A. 1 个　　B. 3 个

C. 5 个　　D. 多个

(2) 如果想实现方法过滤拦截器，则拦截器类需要继承 MethodFilterInterceptor 类，在 MethodFilterInterceptor 类中有两个方法用来指定哪个方法需要拦截，哪个不需要。那么过滤方法的拦截是__________方法。

A. setExcludeMethods　　B. setIncludeMethods

C. doIntercept　　D. 以上方法都不是

(3) 配置默认拦截器使用__________元素，该元素作为<package .../>元素的子元素使用，为该包下的所有 Action 配置默认的拦截器。

A. <default-interceptor-ref …/>

B. <interceptor-ref …/>

C. <interceptor-default-ref …/>

D. <default-interceptor …/>

(4) 下面是配置默认拦截器的配置示例。

```
<package name="包名">
        <!-- 所有拦截器和拦截器栈都配置在该元素下 -->
        <interceptors>
            <!-- 定义拦截器 -->
            <interceptor name="拦截器名" class="拦截器类"/>
            ____(1)_______
        </interceptors>
        ____(2)_______
        <!-- 配置多个 Action -->
        <action --/>
</package>
```

<default-interceptor-ref name="拦截器名"/>应放在上段代码中的__________处。

A. (1)处　　B. (2)处

C. 都可以　　D. 都不可以

三、上机练习

上机练习：为程序添加权限。

要求：做一个简单的登录页面，如图 5-14 所示。对登录用户进行验证，如果是合法用户(合法就是符合标准，标准可以自己定义)可以进入本系统的主页面，如图 5-15 所示，否则无法进入主页面。



图 5-14　登录界面



图 5-15　主页面

进入主页面后，每个人都有查看管理员列表的权限，但非超级管理员无添加、删除管理员的权限，单击“添加管理员”链接或者“删除”链接，转到提示无权限页面，如图 5-16 所示。只有超级管理员登录才有添加、删除管理员的权限，如图 5-17 所示。



图 5-16　非超级管理员无权限



图 5-17　超级管理员添加管理员操作



第 6 章　探索数据校验的奥妙

内容摘要：

在 Web 应用程序中，为了防止客户端传来的数据引发程序的异常，需要对用户输入的数据进行验证。试想一下，如果 Web 应用没有对用户输入数据进行验证，那么当用户由于误操作而输入了一些无效的数据时，程序会向用户显示一大堆的异常栈信息，这是一件多么糟糕的事情。更何况有一些恶意的用户可以通过输入精心伪造的数据来攻击某个系统，破坏系统的运行，窃取协同的机密资料。而这一切，都是因为系统没有对用户输入的数据进行验证。

在 Web 应用程序中构建一个强有力的验证机制，是保障系统稳定运行的前提条件。对用户输入数据进行验证分为两个部分：一是验证输入数据的有效性，二是在用户输入了不正确的数据后向用户提示错误信息。

本章将讲述如何在 Struts 2 中对用户输入数据进行验证。

学习目标：

- 在 Action 中通过编程对输入数据进行验证。
- 掌握 validateXxx 和 validate()方法的使用。
- 熟悉 Struts 2 内置的验证器。
- 掌握验证框架在开发中的使用。
- 熟练开发自定义的验证器。
- 掌握验证注解的使用。

6.1 手动完成输入校验

一个技术成熟的开发者可能已经习惯自己的校验方法，那就是，在 Servlet 中进行有效的服务器端校验。但这种校验很繁琐，需要书写多行的校验代码来完成一个表单的数据校验。而且采用这种校验开发效率低，校验明显不够规范。通常而言，每个 MVC 框架都会提供规范的数据校验部分，Struts 2 也不例外。下面介绍通过手动方式在 Struts 2 中完成输入校验。

 **视频教学：** 光盘/videos/06/validate.avi 长度：9 分钟
光盘/videos/06/validateXxx.avi 长度：7 分钟

6.1.1 基础知识——手动完成输入校验

最直接的验证方式，就是编写 Java 代码对用户提交的数据进行验证。在 Struts 2 中，验证代码是放在 Action 类中完成的。

1. 在Action的execute()方法中进行验证

当一个请求到来的时候，框架调用 Action 的 execute()方法对用户请求进行处理，首先想到的就是在 execute()方法中对用户输入数据的验证。

下面用一个实例来说明问题。

首先在 RegistAction 中编辑 execute 方法，并进行对用户输入数据进行验证，代码如下。

```
public class RegistAction extends ActionSupport {
    //该请求包含的 4 个请求参数
    private String name;
    private String pass;
    private int age;
    private Date birth;
    //在 Action 的 execute 方法中进行验证
    public String execute() throws Exception {
        //如果用户名不为空，且不匹配长度为 4~25 的字母和数字组成的字符串
        if(name!=null&&!Pattern.matches("\\w{4,25}", name.trim())){
            addFieldError("username", "您输入用户名必须是字母和数字，且长度必须是 4
到 25 之间");
            return INPUT;
        }
        //如果密码不为空，且不匹配长度为 4~25 之间的字母和数字组成的字符串
        if(pass!=null&&!Pattern.matches("\\w{4,25}", pass.trim())){
            addFieldError("pass","您输入密码必须是字母和数字,且长度必须是 4 到 25 之间");
            return INPUT;

        }
        //如果年龄不在有效的年龄段内
```

```
        if(age>150 || age<0){
            addFieldError("age", "您输入的年龄必须是一个有效的年龄");
            return INPUT;
        }
        //取得有效生日的最后期限
        Calendar end=Calendar.getInstance();
        end.set(2010, 10,30);
        //取得有效生日的最早期限
        Calendar start=Calendar.getInstance();
        start.set(1900, 1, 1);
        //如果生日不为空，且生日不是一个有效的生日
    if(birth!=null&&(birth.after(end.getTime())||birth.before(start.getTime
()))){
            addFieldError("birth", "您输入的生日必须在一个有效的时间段内");
            return INPUT;
        }
        return SUCCESS;
    }
    /*下面是上面 4 个属性的 set、get 方法，这里省略*/
}
```

当用户请求 RegistAction 中的 execute 方法时，会对输入的数据进行验证，从而判断要返回给客户端的结果页面应该是哪个？“INPUT”指向的结果页面就是用户注册页面 reg.jsp，在本页面中可以用<s:fielderror/>标签输出 Action 中的错误信息，如图 6-1 所示。当用户输入的数据都合法时，进入“SUCCESS”指向的结果页面 success.jsp 页面，如图 6-2 所示。



图 6-1　提示错误信息(注册页面)



图 6-2　注册成功

2．在validateXxx()方法中进行验证

有了上面所讲的在 Action 中的 execute 方法中进行验证做铺垫，下面要讲的在 validateXxx()

方法中进行验证就不难了。



在 execute()方法中对输入数据进行验证是可以工作的，但是它向 execute()方法添加了很多代码。如果一个复杂的表单有很多字段，而且每个字段有多种验证，那么 execute()方法中的代码将急剧膨胀，使得完成业务逻辑的代码淹没在验证代码之中，变得不可辨别。

较上述在 Action 中的 execute()方法中对输入数据进行校验相比，比较好的方式是把验证代码剥离出来，放在一个单独的方法中，然后在 execute()方法中调用该方法。除此之外，也可以利用 Struts 2 提供的便利特性，将验证代码放到形如 validateXxx()的方法中。

在 Struts 2 中，对于多个不同的请求，可以使用同一个 Action 类的不同方法来进行处理，针对特定方法的输入数据的验证处理可以放到 validateXxx()方法中，Xxx 是方法名的首字母大写形式(例如 execute()方法的验证方法为 validateExecute())。不过要注意的是，对于 doXxx()方法，它的验证方法名无须添加 do 前缀，直接写成 validateXxx()即可(例如 doDefault()方法的验证方法为 validateDefault())。

validateXxx()方法由框架自动调用，它将在实际处理请求、实现业务逻辑的方法调用之前被调用。



validateExecute()方法不需要有返回值，如果有验证错误，直接将它添加到 Action 的字段错误中即可。validateXxx()方法是由 com.open symphony.xwork2.interceptor. DefaultWorkFlowInterceptor 拦截器(已包含在 defaultStack 拦截器栈中)调用的。

3. 在validate()方法中进行验证

如果处理用户请求的多个 Action 方法的验证逻辑是相同的，那么可以让 Action 类实现 com.opensymphony.xwork2.Validateable 接口，然后在该接口定义的 validate()方法中对用户输入数据进行验证。不管处理请求的 Action 方法是哪一个，validate()方法都会被调用。

validate()方法是由 DefaultWorkFlowInterceptor 拦截器调用的，DefaultWorkFlowInterceptor 会检查 Action 是否实现了 Validateable 接口，如果实现了，就调用该接口中的 validate()方法。validate()方法在 validateXxx()方法调用之后被调用，并且无论 validateXxx()方法执行结果如何，validate()方法都会被调用。

调用 validate()方法的拦截器该拦截器 DefaultWorkFlowInterceptor 有一个 excludeMethods 参数，用于指定要排除的方法，对于被排除的方法，DefaultWorkFlowInterceptor 拦截器的 doIntercept()方法将不会被调用，因此 validate()方法也就不会被调用了。

struts-default.xml 文件中，DefaultWorkFlowInterceptor 拦截器的配置(在 defaultStack 拦截器栈的配置中)如下。

```
<interceptor-ref name="workflow">
        <param name="excludeMethods">input,back,cancel</param>
</interceptor-ref>
```

从这个配置中读者可以了解到，将一个表单的 Action 方法命名为 input、back、cancel 或者

doInput、doBack、doCancel，那么 validate()方法将不会执行。如果需要配置拦截器要排除的方法，需要修改 struts.xml 文件，如下所示。

```
<interceptor-ref name="defaultStack">
        <param name="workflow.excludeMethods">xxx</param>
</interceptor-ref>
```

在 Action 中的 execute()方法中对输入数据进行校验的代码，一旦发现校验失败，就把校验失败提示通过 addFieldError 方法添加进系统的 fieldError 中，在这里的 validate()方法中也一样。

为了在 input 视图对应的 JSP 页面中输出错误提示，应该在该页面中增加如下代码。

```
<!--校验失败提示信息-->
<s:fielderror/>
```

上面的<s:fielderror/>标签专门负责输出系统的 fieldError 信息，也就是输出系统的输入校验失败提示。



即使类型转换失败，系统也不会直接返回 input 逻辑视图，依然会调用 Action 的 validate 方法来进行输入校验。这一点，与前面介绍的类型转换失败处理后的处理时一致的。

4. Struts 2 的输入校验流程

通过上面的详细介绍，不难发现 Struts 2 的输入校验需要经过如下几个步骤。

(1) 类型转换器负责对字符串的请求参数执行类型转换，并将这些值设置成 Action 的属性值。

(2) 在执行类型转换过程中可能出现异常，如果出现异常，将异常信息保存到 ActionContext 中，conversionError 拦截器负责将其封装到 fieldError 里，然后执行第(3)步；如果转换过程没有异常信息，则直接进入第(3)步。

(3) 通过反射调用 validateXxx()方法，其中 Xxx 是即将要处理用户请求的处理逻辑所对应的方法名。

(4) 调用 Action 类里的 validate()方法。

(5) 如果经过上面 4 步都没有出现 fieldError，将调用 Action 里处理用户请求的处理方法；如果出现了 filedError，系统将转入 input 逻辑视图所指定的视图资源。

不喜欢看文字的读者，可以看下图 6-3 分解。

6.1.2 实例描述

输入校验是一项常用的技术，通过这种技术可以校正用户的输入，从而可以避免一些不必要的错误。随着数字化时代的到来，这种校验已经随处可见，而在 Web 应用当中尤为常见，随处可以看到它的身影。本案例我们来制作一个关于登录的输入验证。在该实例中，如果用户输入的用户名为特殊字符，那么将会出现出错提示，并提示用户输出字符、数字或者两者的组合。具体实现方法如下。

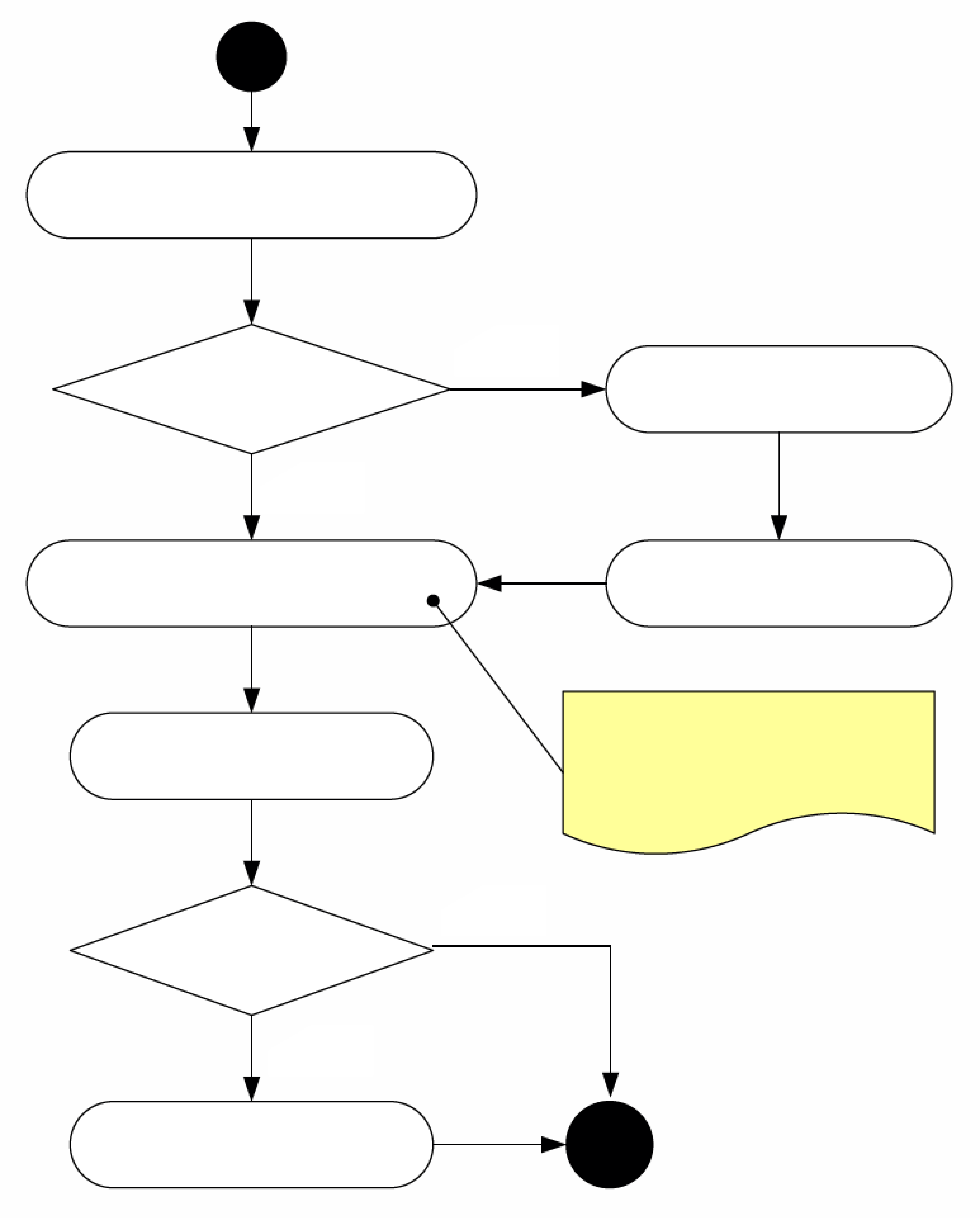


图 6-3　validate()方法校验顺序图

6.1.3　实例应用

【例 6-1】 验证用户输入是否合法。

(1) 配置 web.xml(配置 FilterDispatcher)。

(2) 创建实体类 User.java，里面有两个属性，一个是 username，一个是 password，代码如下。

```
package com.struts2.model;
/**
*用户实体类
*@author Administrator
*
*/
public class User {
    private String username;//用户名
```

```
    private String password;//密码
    /*下面是上面属性 username、password 的 set、get 方法，这里省略*/
}
```

(3) 创建登录 Action，名为 LoginAction，在其中编辑 execute()方法，把登录用户对象放入 Request 对象中，同时编辑它的 validateXxx()方法，对用户输入数据进行判断，代码如下。

```
package com.struts2.actions;
import java.util.regex.Pattern;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts2.ServletActionContext;
import com.opensymphony.xwork2.ActionSupport;
import com.struts2.model.User;
public class LoginAction extends ActionSupport {
    private User user;
    @Override
    public String execute() throws Exception {
        //获取 HttpServletRequest 对象
        HttpServletRequest request=ServletActionContext.getRequest();
        //把从表单获取到的 User 对象存放到 HttpServletRequest 对象中
        request.setAttribute("login", user);
        return "loginSuccess";
    }
    /**
    * execute()的 validateXxx 方法，用于执行 execute()之前
    */
    public void validateExecute(){
        //获取表单数据 username
        String uname=user.getUsername();
        //判断 username 是否合法
        if(uname==null||!Pattern.matches("\\w{4,25}",uname.trim())){
            addFieldError("user.username", "您输入的用户名必须是字母和数字，且长度
必须在 4 到 25 之间！");
        }
        //获取表单数据 password
        String pwd=user.getPassword();
        //判断 password 是否合法
        if(pwd==null||!Pattern.matches("\\w{4,25}", pwd.trim())){
            addFieldError("user.password", "您输入的密码必须是字母和数字，且长度必
须在 4 到 25 之间！");
        }
    }
    /*下面是上面 User 对象的 set、get 方法，这里省略*/
}
```

(4) 在 src 目录下的 struts-config 文件夹中新建 login.xml 文件，配置 LoginAction，配置如下代码。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
```

```
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name ="login" extends ="struts-default" >

        <action name ="login" class ="com.struts2.actions.LoginAction" >
            <result name="loginSuccess">/main.jsp</result>
            <result name="input">/login.jsp</result>
        </action>
    </package >
</struts>
```

(5) 在 WebRoot 文件夹下新建 login.jsp 页面作为登录页面，代码片段如下。

```
<%@taglib prefix="s" uri="/struts-tags" %>
<form name=form1 action="login/login.action" method=post>
    <input type="text" name="user.username" value="<s:property
value="user.username"/>"/>
    <input type="text" name="user.password" value="<s:property
value="user.password"/>"/>
    <input type="submit" value="登录"/>
    <s:fielderror/>
</form>
```

(6) 在 src 目录下的 struts.xml 文件中引入 login.xml 文件。

6.1.4 运行结果

运行程序中的 login.jsp 页面，在用户名输入框中输入非字母、非数字，无法登录到主页面，并且提示错误信息，如图 6-4 所示。



图 6-4 输入验证，提示错误信息

6.1.5　实例分析



源码解析：

上述案例中，在 LoginAction 中的 validateExecute()方法中验证用户输入时并未返回任何结果，在它的 execute()方法中只是返回了一个“loginSuccess”的结果页面 main.jsp。但是系统在运行时，进入 validateExecute()方法后，当验证用户输入不合法时跳转到了 login.jsp 页面，这是为什么呢？

当系统验证用户输入不合法时，会自定跳转至“INPUT”所对应的结果页面，而我们在 login.xml 文件中配置了 input 所对应的结果页面正是 login.jsp，因此会跳转至登录页面 login.jsp，并以<s:fielderror/>标签显示错误信息。

6.2　基本输入校验

在上一节中介绍了在 Action 类中编写代码对用户输入数据进行校验，然而，这种验证方式也存在着一些问题。首先是当验证规则比较复杂时，Action 类的代码将变得繁杂不堪；其次是某些验证规则无法复用，例如判断字段是否为空、判断字符串的长度等规则，对于很多字段来说都是需要的。在上一节中，仅仅只是完成了对必填字段是否合法的判断，对于字段的格式并没有做详细的验证，编写这样的验证代码费时费力，使得代码的编写变成了体力活。

正因为输入验证的重要性和重复性，才有了一些验证框架的推出。Struts 2 也内置了一个验证框架，开发者可以通过在外部配置文件中定义验证规则的方式来简化对输入数据的验证，从而可以减轻开发者的负担，提高开发效率。

下面将给读者介绍一下如何编写一个校验规则文件来对输入数据进行校验，校验规则文件如何获取国际化文件中的 key，并提示错误信息及校验文件的搜索规则是怎样的？



视频教学：光盘/videos/06/registValidate.avi　　　　**长度：**7 分钟

6.2.1　基础知识——基本输入校验

6.1 节介绍的手动校验方式，虽然比之前的输入校验有一定的简化，但依然需要手动编写大量的代码，编程依然很繁琐，代码复用性也很低。Struts 2 提供了基于验证框架的输入校验，在这种校验方式下，所有的输入校验只需要通过指定简单的配置文件即可。

1．编写校验规则文件

讲大多的文字不如用一个实例来帮助理解直观，下面将用一个已经用到过的实例，来给读者讲解一下如何用一个 xml 文件配置来验证用户输入信息。

将上面的 RegistAction 类的 execute()方法中验证用户输入的信息删除，只返回一个“SUCCESS”的结果页面，修改后的代码如下。

```
public class RegistAction extends ActionSupport {
    //该请求包含的 4 个请求参数
    private String name;
    private String pass;
    private int age;
    private Date birth;
    //在 Action 的 execute 方法中进行验证
    public String execute() throws Exception {
        return SUCCESS;
    }
    /*下面是上面 4 个属性的 set、get 方法，这里省略*/
}
```

在上面的 Action 中，提供了四个属性类封装用户的请求参数，并为这四个参数提供了对应的 setter 和 getter 方法。初看起来，这个类是一个普通 JavaBean，不是 Action，但因为它继承了 ActionSupport 类，因此它包含了一个 execute()方法，且该方法直接返回 success 字符串。

采用 Struts 2 的校验框架时，只需要为该 Action 指定一个校验文件即可。校验文件是一个 XML 配置文件，该文件指定了 Action 的属性必须满足特殊的规则，下面是该应用中 Action 的校验文件的代码。

```
<?xml version="1.0" encoding="GBK"?>
<!-- 指定校验配置文件的 DTD 信息 -->
<!DOCTYPE validators PUBLIC
 "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
 "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<!-- 校验文件的根元素 -->
<validators>
    <!-- 校验 Action 的 name 属性 -->
    <field name="name">
        <!-- 指定 name 属性必须满足必填规则 -->
        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <message>必须输入名字</message>
        </field-validator>
        <!-- 指定 name 属性必须匹配正则表达式 -->
        <field-validator type="regex">
            <param name="expression"><![CDATA[(\w{4,25})]]></param>
            <message>您输入的用户名只能是字母和数字,且长度必须在 4 到 25 之间</message>
        </field-validator>
    </field>
    <!-- 校验 Action 的 pass 属性 -->
    <field name="pass">
        <!-- 指定 pass 属性必须满足必填规则 -->
        <field-validator type="requiredstring">
            <param name="trim">true</param>
```

```
            <message>必须输入密码</message>
        </field-validator>
        <!-- 指定 pass 属性必须匹配正则表达式 -->
        <field-validator type="regex">
            <param name="expression"><![CDATA[(\w{4,25})]]></param>
            <message>您输入的密码只能是字母和数字，且长度必须在 4 到 25 之间</message>
        </field-validator>
    </field>
    <!-- 指定 age 属性必须在制定范围内 -->
    <field name="age">
        <field-validator type="int">
            <param name="min">1</param>
            <param name="max">150</param>
            <message>年龄必须在 1 到 150 之间</message>
        </field-validator>
    </field>
    <!--指定 birth 属性必须在指定范围内 -->
    <field name="birth">
        <field-validator type="date">
            <!-- 下面指定日期字符串时，必须使用本 Locale 的日期格式 -->
            <param name="min">1900-01-01</param>
            <param name="max">2010-02-21</param>
            <message>年龄必须在${min}到${max}之间</message>
        </field-validator>
    </field>
</validators>
```

Struts 2 的校验文件规则与 Struts 1 的校验文件设计方式不同，Struts 2 中每个 Action 都有一个校验文件，而且这个校验文件的命名有一定的规范性，大致可以分为两种。

- Action 级别校验命名格式：

```
ActionClassName-validation.xml
```

- Action 中某个方法的校验命名格式：

```
ActionClassName-ActionAliasName-validation.xml
```



这里的 ActionAliasName(Action 别名)指的是在 struts.xml 文件中配置的 Action name= “xxx” 的 xxx 名称，而不是 method= “xxx” 的名称。

前面的 ActionClassName 和 ActionAliasName 是可以改变的，后面的-validation.xml 部分总是固定的，且该文件应该被保存在与 Action class 文件相同的路径下。例如，本应用的 Action class 文件保存在 WEB-INF/classes/com/struts2/actions 路径下，故该校验文件也应该保存在该路径下。

增加了该校验文件后，其他部分无需任何修改，系统自动会加载该文件，当用户提交请求时，Struts 2 的校验框架会根据该文件对用户请求进行校验。如果用户输入不满足校验规则，将可以看到如图 6-5 所示的界面。



图 6-5　使用校验框架提示验证信息

从上图中可以看出，这种基于 Struts 2 校验框架的校验方式完全可以替代手动校验，而且这种校验方式的可重用性非常高，只需要在配置文件中配置校验规则，即可完成数据校验，无需用户书写任何的数据校验代码。

2. 国际化提示信息

在上面的数据校验中，所有的提示信息都是用编码的方式写在配置文件中，这种方式显然不利于程序国际化。

1）　通过 key 指定国际化提示信息

当查看每个校验文件时，发现每个<field-validator.../>元素都包含了一个必填的<message.../>子元素，这个子元素中的内容就是校验失败后的提示信息。为了国际化该提示信息，为 message 元素指定 key 属性，该 key 属性指定是国际化提示信息对应的 key。

下面是改写后的校验规则文件代码，命名为 RegistAction-reg-validation.xml。

```
<validators>
    <!-- 校验 Action 的 name 属性 -->
    <field name="name">
        <!-- 指定 name 属性必须满足必填规则 -->
        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <message key="name.requried"/>
        </field-validator>
        <!-- 指定 name 属性必须匹配正则表达式 -->
        <field-validator type="regex">
            <param name="expression"><![CDATA[(\w{4,25})]]></param>
            <message key="name.regex"/>
        </field-validator>
    </field>
```

```
    <!-- 校验 Action 的 pass 属性 -->
    <field name="pass">
        <!-- 指定 pass 属性必须满足必填规则 -->
        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <message key="pass.requried"/>
        </field-validator>
        <!-- 指定 pass 属性必须匹配正则表达式 -->
        <field-validator type="regex">
            <param name="expression"><![CDATA[(\w{4,25})]]></param>
            <message key="pass.regex"/>
        </field-validator>
    </field>
    <!-- 指定 age 属性必须在制定范围内 -->
    <field name="age">
        <field-validator type="int">
            <param name="min">1</param>
            <param name="max">150</param>
            <message key="age.range"/>
        </field-validator>
    </field>
    <!--指定 birth 属性必须在指定范围内 -->
    <field name="birth">
        <field-validator type="date">
            <!-- 下面指定日期字符串时，必须使用本 Locale 的日期格式 -->
            <param name="min">1900-01-01</param>
            <param name="max">2010-02-21</param>
            <message key="birth.range"/>
        </field-validator>
    </field>
</validators>
```

从上面校验规则文件中可以看出，没有直接给校验后的提示信息，每个 message 元素只是指定了一个 key 属性，该 key 属性就是校验失败后给出的国际化提示信息对应的 key。

因为上面的校验文件中指定了许多国际化信息的 key，所以必须在国际化资源文件中增加对应的 key，即在 RegistAction 所在的同目录下新建一个 RegistAction.properties 文件，增加如下 Entry。

```
#违反用户名必须输入的提示信息
name.requried=您必须输入用户名!
#违反用户名必须匹配正则表达式的提示信息
name.regex=您输入的用户名只能是字母和数字，且长度必须在 4 到 25 之间!
#违反密码必须输入的提示信息
pass.requried=您必须输入密码!
#违反密码必须匹配正则表达式的提示信息
pass.regex=您输入的密码只能是字母和数字，且长度必须在 4 到 25 之间!
#违反年龄必须在指定范围的提示信息
age.range=您的年龄必须在${min}和${max}之间!
```

```
#违反生日必须在指定范围的提示信息
birth.range=您的生日必须在${min}和${max}之间!
```

接下来，我们来看看运行结果是否和上一节运行结果一样！运行 reg.jsp 页面，输入一个年龄为 190，单击 Regist 按钮，出现如图 6-6 所示的界面。



图 6-6　国际化提示信息校验输入数据

通过上面的讲解，你是否觉得 Struts 2 真的很伟大呢？它不仅提供了<message key="国际化消息"/>的元素来调用国际化，还提供了另一种形式，下面就来为读者讲解一下 Struts 2 提供的另一种形式调用国际化资源文件提示信息。

2)　通过 ActionSupport 的 getText()方法获取国际化提示信息

首先将输入页面的表单元素改为使用 Struts 2 标签来生成表单，并且为该表单增加 validate="true"属性即可。

将上面的 reg.jsp 页面的代码改为如下代码。

```
<s:form action="reg/reg.action" method="post" validate="true">
        <!-- 使用 s:textfield 标签生成文本输入框 -->
        <s:textfield label="username" name="name"/>
        <s:password label="password" name="pass"/>
        <s:textfield label="user age" name="age"/>
        <s:textfield label="birthday" name="birth"/>
        <s:submit value="Regist"/>
</s:form>
```

根据 Struts 2 的官方文档所说的，只要将 JSP 页面改成如上的形式，即应该可以完成客户端校验。为了能满足用户的编码爱好，Struts 2 还提供了另一种方式来输出国际化资源文件：${getText("消息 key")}，于是将校验文件改为如下形式。

```
<validators>
    <!-- 校验 Action 的 name 属性 -->
```

```
    <field name="name">
        <!-- 指定 name 属性必须满足必填规则 -->
        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <message>${getText("name.requried")}</message>
        </field-validator>
        <!-- 指定 name 属性必须匹配正则表达式 -->
        <field-validator type="regex">
            <param name="expression"><![CDATA[(\w{4,25})]]></param>
            <message>${getText("name.regex")}</message>
        </field-validator>
    </field>
    <!-- 校验 Action 的 pass 属性 -->
    <field name="pass">
        <!-- 指定 pass 属性必须满足必填规则 -->
        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <message>${getText("pass.requried")}</message>
        </field-validator>
        <!-- 指定 pass 属性必须匹配正则表达式 -->
        <field-validator type="regex">
            <param name="expression"><![CDATA[(\w{4,25})]]></param>
            <message>${getText("pass.regex")}</message>
        </field-validator>
    </field>
    <!-- 指定 age 属性必须在制定范围内 -->
    <field name="age">
        <field-validator type="int">
            <param name="min">1</param>
            <param name="max">150</param>
            <message>${getText("age.range")}</message>
        </field-validator>
    </field>
    <!--指定 birth 属性必须在指定范围内 -->
    <field name="birth">
        <field-validator type="date">
            <!-- 下面指定日期字符串时，必须使用本 Locale 的日期格式 -->
            <param name="min">1900-01-01</param>
            <param name="max">2010-02-21</param>
            <message>${getText("birth.range")}</message>
        </field-validator>
    </field>
</validators>
```

在上面的校验规则文件中，也没有直接校验失败的提示信息，而是通过调用 ActionSupport 的 getText()方法来取得国际化的提示信息。这种配置方式比前面的配置文件要繁琐一点，但对于需要使用客户端校验的情况，还是建议使用这种配置方式。

此时，如果浏览者的输入不再符合校验规则，将看到如图 6-7 所示的界面。



图 6-7　客户端校验效果

从上图 6-7 中可以看出，尽管使用客户端校验，却看不到弹出 JavaScript 的警告框，这种效果看起来与服务器端校验几乎完全相同。



客户端校验依然是基于 JavaScript 完成的，由于 JavaScript 脚本本身的限制，有些服务器端校验不能转换成客户端校验。也就是说，并不是所有的服务器端校验都可以转换成客户端校验的。

客户端校验仅仅支持如下几种校验器。

- required validator (必填校验器)
- requiredstring validator (必填字符串校验器)
- stringlength validator (字符串长度校验器)
- regex validator (表达式校验器)
- email validator (邮件校验器)
- url validator (网址校验器)
- int validator (整数校验器)
- double validator (双精度数字校验器)

客户端校验有两个值得注意的地方。

- Struts 2 的<s:form…/>元素有一个 theme 属性，不要将该属性指定为 simple。
- 不要在校验规则文件的错误提示信息中，直接使用 key 来指定国际化提示信息。

3．校验文件的搜索规则

Struts 2 的一个 Action 中可能包含多个处理逻辑，当一个 Action 类中包含多个类似于 execute 的方法时，每个方法都是一个处理逻辑。不同的处理逻辑可能需要不同的校验规则，Struts 2 也提供了不同 Action 指定不同校验规则的支持。

当需要让一个 Action 可以处理多个请求时，应该在配置该 Action 时指定 method 属性。通过这种方式，就可以将一个 Action 处理类配置成多个逻辑 Action。

在上面的 Action 类中增加一个 login 方法，该 login 方法不做任何处理，只是简单地返回

succcess 字符串。下面在 src 目录下 struts-config 文件夹下的 reg.xml 文件中将该 Action 类配置成两个逻辑 Action。下面是配置这两个逻辑 Action 的配置代码。

```
<package name ="reg" extends ="struts-default" >
        <!-- 配置一个名为 reg 的 Action，对应的处理逻辑为 RegistAction 的 execute 方法
-->
        <action name ="reg" class ="com.Struts 2.actions.RegistAction" >
            <result name="input">/reg.jsp</result>
            <result name="success">/success.jsp</result>
        </action >
        <!-- 配置一个名为 userLogin 的 Action，对应的处理逻辑为 RegistAction 的 login
方法 -->
        <action name="userLogin" class="com.struts2.actions.RegistAction">
            <result name="input">/reg.jsp</result>
            <result name="success">/success.jsp</result>
        </action>
</package >
```

假设上面两个 Action 的校验规则不同，注册时的校验规则还是之前的校验规则，但登录的校验规则需要增加一些校验。如果按之之前的方式来指定校验规则文件，这个校验规则文件肯定分不清楚到底要校验哪个处理逻辑。为了能精确控制每个校验逻辑，Struts 2 允许通过为校验规则文件名增加 Action 别名来指定具体需要校验的处理逻辑。即就是采用前面提到过的形式。

```
<ActionClassName>-<ActionAliasName>-validation.xml
```

例如，需要为 login 处理逻辑单独指定校验规则，则校验文件的文件名为：RegistAction-login-validation.xml(该文件也需要与 RegistAction 的 class 文件放在同一路径下)。该文件的内容片段如下。

```
<validators>
    <!-- 校验 Action 的 name 属性 -->
    <field name="name">
        <!-- 指定 name 属性必须和密码相同 -->
        <field-validator type="fieldexpression">
            <param name="expression"><![CDATA[(name == pass)]]></param>
            <message>${getText("namepass")}</message>
        </field-validator>
    </field>
</validators>
```

在 reg.jsp 页面中添加一个 value=“登录”的按钮作为提交按钮，并修改前面的注册校验规则文件名为 RegistAction-validation.xml，同时要修改 reg.jsp 页面中的 <s:form action=“reg/userLogin.action”…/>。运行 reg.jsp 页面，单击“登录”按钮，将看到如图 6-8 所示的页面。

从运行结果来看，RegistAction-validation.xml 文件中的校验规则，依然会对名为 userLogin 的 Action 起作用。实际上，名为 userLogin 的 Action 中包含的校验规则是 RegistAction-validation.xml 和 RegistAction-userLogin-validation.xml 两个文件中规则的总和。



图 6-8　增加校验规则

除此之外，还有一种情形——系统中包含了两个 Action：BaseAction 和 RegistAction，其中 RegistAction 继承了 BaseAction，且两个 Action 都指定了对应的配置文件，则 RegistAction 对应 Action 的校验规则实际上是 RegistAction-validation.xml 和 BaseAction-validation.xml 两个文件规则的总和。

假设系统有两个 Action：BaseAction 和 RegistAction，则系统搜寻规则文件顺序如下。

(1) BaseAction-validation.xml

(2) BaseAction-别名-validation.xml

(3) RegistAction-validation.xml

(4) RegistAction-别名-validation.xml

这种搜寻与其他搜寻不同的是，即使找到第一个校验规则，系统还会继续搜索，不管有没有这四份文件，也不管是否找到匹配文件，系统总是按固定顺序搜索。



Struts 2 搜索规则文件是从上而下，实际用的校验规则是所有校验规则的总和。如果两个校验文件中指定的校验规则冲突，则从后面文件中的校验规则取值。

6.2.2　实例描述

今天，我正在吃午饭，客户一个电话让我不得不放下饭碗跑回公司。是什么让我这么紧张呢？我的天呢！客户要修改功能，并且修改的功能还不少呢！这个程序是我们以前的开发者做的，也不知怎么想的，对那些用户输入数据进行校验，全部都是在 Action 中使用 validateXxx()方法进行校验。现在客户要修改功能，需要添加字段，就意味着要对新增字段进行数据校验，要修改 Action 吗？不会吧！修改的功能可不少啊，这要一个 Action 一个 Action 的修改吗？于是，我从下午忙到第二天晚上才把事情做完时，气的客户脸都绿了。殊不知不怨我，要是这个系统把用户输入数据的验证用一个校验规则文件写，那修改起来多方便啊！

下面我将把系统中的注册功能给读者分享一下。

6.2.3 实例应用

【例 6-2】 对用户注册时的输入数据进行验证。

(1) 修改用户信息实体类 User.java，新增属性 email 和 age，修改后的代码如下。

```
public class User {
    private String username;//用户名
    private String password;//密码
    private String email;//邮箱地址
    private int age;//年龄
    /*下面是上面所有属性的 get、set 方法，这里省略*/
}
```

(2) 新建用户注册 Action，命名为 UserRegistAction，继承 ActionSupport，并重写 execute()方法，在 execute()方法中无任何操作，只是返回结果页面。UserRegistAction 的内容如下。

```
public class UserRegistAction extends ActionSupport {
    private User user;
    private String verifyPassword;//确认密码
    @Override
    public String execute() throws Exception {
        return SUCCESS;
    }
    /*下面是上面两个属性的 get、set 方法，这里省略*/
}
```

(3) 在 struts-config 文件夹下新建 userRegist.xml 文件，配置 UserRegistAction 类，代码如下。

```
<package name ="userRegist" extends ="struts-default" >
        <action name="userRegist"
class="com.struts2.actions.UserRegistAction">
            <result name="success">/success.jsp</result>
            <result name="input">/userRegist.jsp</result>
        </action>
</package >
```

(4) 在 WebRoot 下新建 userRegist.jsp 页面作为注册页面，提交至 userRegist/userRegist.action，代码如下。

```
<s:form action="userRegist/userRegist.action" method="post" validate="true">
            <s:textfield name="user.username" label="用户名" required="true"/>
            <s:password name="user.password" label="密码" required="true"/>
            <s:password name="verifyPassword" label="确认密码"
required="true"/>
            <s:textfield name="user.age" label="年龄" value="10"
required="true"/>
            <s:textfield name="user.email" label="邮箱" required="true"/>
            <s:submit value="注册"/>
</s:form>
```

(5) 到目前为止，已可以运行 userRegist.jsp 页面，无论输入什么，都可以转到“SUCCESS”所对应的成功页面 success.jsp。但这样有可能会使系统崩溃。因此必须对用户的输入数据做一些校验。编辑校验规则文件 UserRegistAction-validation.xml，对用户输入的所有数据进行校验，校验规则文件内容如下。

```
<validators>
    <!-- 针对 user.username 字段的验证规则 -->
    <field name="user.username">
        <!-- 使用 requiredstring 验证器，确保 user.username 字段值不为 null，也不为""
-->
        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <message key="error.username.required"/>
        </field-validator>
        <!-- 使用 stringlength 验证器，确保 user.username 字段值的字符长度在 4 到 12 之
间 -->
        <field-validator type="stringlength">
            <param name="minLength">4</param>
            <param name="maxLength">12</param>
            <message key="error.username.length"/>
        </field-validator>
    </field>
    <!-- 针对 user.password 字段的验证规则 -->
    <field name="user.password">
        <field-validator type="requiredstring">
            <message key="error.password.required"/>
        </field-validator>
        <field-validator type="stringlength">
            <param name="minLength">4</param>
            <param name="maxLength">8</param>
            <message key="error.password.length"/>
        </field-validator>
    </field>
    <!-- 确认密码字段的验证规则 -->
    <field name="verifyPassword">
        <field-validator type="requiredstring">
            <message key="error.verifyPassword.required"/>
        </field-validator>
        <!-- 使用 fieldexpression 验证器 -->
        <field-validator type="fieldexpression">
            <!-- 使用 OGNL 表达式来判断确认密码和密码的一致性 -->
            <param name="expression">verifyPassword==user.password</param>
            <message key="error.verifyPassword.identical"/>
        </field-validator>
    </field>
    <!-- 针对 user.age 字段的验证规则 -->
    <field name="user.age">
        <field-validator type="int">
            <param name="minLength">10</param>
            <param name="maxLength">100</param>
```

```
            <message key="error.age.invalid"/>
        </field-validator>
    </field>
    <!-- 针对 user.email 字段的验证规则 -->
    <field name="user.email">
        <field-validator type="requiredstring">
            <message key="error.email.required"/>
        </field-validator>
        <!-- 使用 email 验证器，确保 user.email 字段值是合法的邮件地址 -->
        <field-validator type="email">
            <message key="error.email.invalid"/>
        </field-validator>
    </field>
</validators>
```

校验规则文件 UserRegistAction-validation.xml 与 UserRegistAction 的 class 位于同一目录下，这里都位于 com/struts2/actions 路径下。

(6)　编辑国际化文件 UserRegistAction.properties，内容如下。

```
error.username.required=请输入用户名!
error.username.length=用户名长度必须在${minLength}到 ${maxLength}之间!
error.password.required=请输入密码
error.password.length=密码长度必须在${minLength}到${maxLength}之间!
error.age.invalid=年龄必须在${minLength}到${maxLength}之间!
error.email.required=请输入邮箱!
error.email.invalid=您的邮箱地址无效!
verifyPassword=确认密码!
error.verifyPassword.required=请再次输入密码!
error.verifyPassword.identical=两次输入的密码不一致!
```

6.2.4　运行结果

运行 userRegist.jsp 页面，如果输入错误，出现如图 6-9 所示的界面。

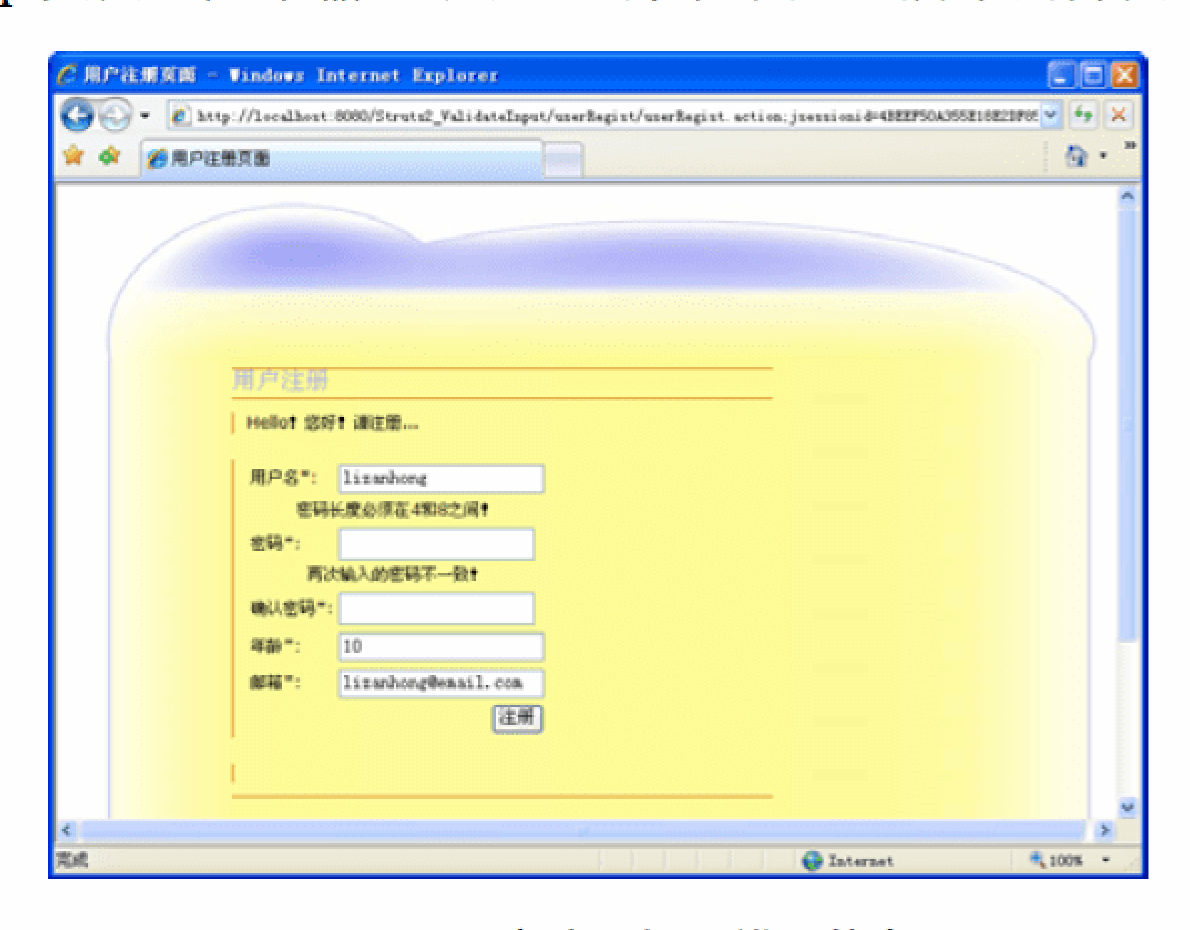


图 6-9　用户注册提示错误信息

6.2.5 实例分析


源码解析：

上案例中，我们对确认密码进行了验证，运用 type="fieldexpression"的 field-validator 元素，同时联合<param name="expression">verifyPassword==user.password</param>要求两次输入的密码要一致，其中 verifyPassword 指的是在 Action 中的属性，fieldexpression validator 是字段表达式验证器。使用邮箱地址验证器(email validator)对用户输入的邮箱地址进行了验证，无需写正则表达式。验证字符串是否是合法邮件地址的正则表达式已经内置在 EmailValidator 验证器中。

6.3 内置校验器

就如拦截器一样，有自定义的当然也有内置(内建)的拦截器，这样才能使 Struts 2 框架更加的完美。

下面就为读者讲解一下 Struts 2 内置的校验器有哪些，以及每个检验器都具有什么样的功能，在何种情况下使用。

视频教学： 光盘/videos/06/SystemValidate.avi 长度：12 分钟
光盘/videos/06/conversion.avi 长度：8 分钟
光盘/videos/06/date.avi 长度：7 分钟
光盘/videos/06/email.avi 长度：7 分钟
光盘/videos/06/expression.avi 长度：10 分钟
光盘/videos/06/int.avi 长度：10 分钟
光盘/videos/06/othervalidate.avi 长度：6 分钟
光盘/videos/06/requiredstring.avi 长度：7 分钟
光盘/videos/06/stringlength.avi 长度：7 分钟

6.3.1 基础知识——内置校验器

Struts 2 提供了大量的内置校验器，这些内置的校验器可以满足大部分应用的校验需求，开发者只需要使用这些校验器即可。

Struts 2 针对常用的验证器需求，提供了 13 个验证器。使用 WinRAR 打开 Struts 2 发布包的解压缩文件中的 xwork-core-2.1.6.jar 文件，在该压缩文件的 com/opensymphony/xwork2/validator/validators 路径下找到一个 default.xml 文件，这个文件就是 Struts 2 默认的校验器注册文件。该文件的代码如下。

```
    <validator name="required"
class="com.opensymphony.xwork2.validator.validators.RequiredFieldValidator"/>
```

```
    <validator name="requiredstring"
class="com.opensymphony.xwork2.validator.validators.RequiredStringValidator
"/>
    <validator name="int"
class="com.opensymphony.xwork2.validator.validators.IntRangeFieldValidator"/>
    <validator name="long"
class="com.opensymphony.xwork2.validator.validators.LongRangeFieldValidator"/>
    <validator name="short"
class="com.opensymphony.xwork2.validator.validators.ShortRangeFieldValidator"/>
    <validator name="double"
class="com.opensymphony.xwork2.validator.validators.
DoubleRangeFieldValidator"/>
    <validator name="date"
class="com.opensymphony.xwork2.validator.validators.DateRangeFieldValidator"/>
    <validator name="expression"
class="com.opensymphony.xwork2.validator.validators.ExpressionValidator"/>
    <validator name="fieldexpression"
class="com.opensymphony.xwork2.validator.validators.
FieldExpressionValidator"/>
    <validator name="email"
class="com.opensymphony.xwork2.validator.validators.EmailValidator"/>
<validator name="url"
class="com.opensymphony.xwork2.validator.validators.URLValidator"/>
    <validator name="visitor"
class="com.opensymphony.xwork2.validator.validators.VisitorFieldValidator"/>
    <validator name="conversion"
class="com.opensymphony.xwork2.validator.validators.
ConversionErrorFieldValidator"/>
    <validator name="stringlength"
class="com.opensymphony.xwork2.validator.validators.
StringLengthFieldValidator"/>
    <validator name="regex"
class="com.opensymphony.xwork2.validator.validators.RegexFieldValidator"/>
    <validator name="conditionalvisitor"
class="com.opensymphony.xwork2.validator.validators.
ConditionalVisitorFieldValidator"/>
</validators>
```

上面代码中注册的校验器，就是 Struts 2 全部的内置校验器。

通过上面代码可以看出，注册一个校验器是如此简单：通过一个<validator.../>元素即可注册一个校验器，每个<validator.../>元素的 name 属性指定该校验器的名字，class 属性指定该校验器的实现类。

如果开发者开发了一个子集的校验器，则可以通过添加一个 validation.xml 文件(该文件应该放在 WEB-INF/classes 路径下)来注册校验器。validation.xml 文件的内容也是由多个<validator .../>元素组成，每个<validator .../>元素注册一个校验器。



如果 Struts 2 系统在 WEB-INF/classes 路径下找到一个 validators.xml 文件，则不会再加载系统默认的 default.xml 文件。因此，如果开发者提供了自己的校验器注册文件(validators.xml 文件)，一定要把 default.xml 文件里的全部内容复制到 validators.xml 文件中。但是也曾经有人证实的结果却与此相反：即使没有在 validators.xml 文件里注册，那些内置的验证程序也可以使用它们。我建议大家为了安全起见，还是在 validators.xml 文件中注册一下吧！

下面将为读者介绍一下 Struts 2 内置的这 13 个验证器的使用情况。

1. 必填验证器(required validator)

RequiredFieldValidator 验证器检查指定的字段是否为 null。该验证器可以接受一个参数：fieldName，指定要验证的字段名。如果使用<field>元素来声明该字段验证器，则不需要这个参数。

采用非字段校验器配置风格时，该校验器的配置示例如下。

```
<validators>
    <!-- 使用非字段校验器风格来配置必填校验器-->
    <!-- 使用 type 属性指定使用 required 验证器 -->
    <validator type="required">
        <!-- 通过 fieldName 参数来指定要验证的字段 -->
        <param name="fieldName">user.username</param>
        <message>请输入用户名! </message>
    </validator>
</validators>
```

采用字段校验器配置风格时，该校验器的配置示例如下。

```
<validators>
    <!-- 使用字段校验器风格来配置必填校验器，校验 user.username 属性-->
    <!-- 通过 name 属性指定要验证的字段名 -->
    <field name="user.username">
        <!-- 使用 type 属性指定使用 required 验证器 -->
        <field-validator type="required">
            <message>请输入用户名! </message>
        </field-validator>
    </field>
</validators>
```

2. 必填字符串验证器(requiredstring validator)

RequiredString Validator 验证器检查一个字符串字段值是否为 null，并且其长度大于 0(即不为“”)。该验证器可以接受两个参数：

- fieldname：指定要验证的字段名。如果使用<field>元素来声明该字段验证器，则不需要这个参数。
- trim：布尔值，指定在执行长度检测之前是否调用 String 的 trim()方法删除首尾的空格。默认值为 true。

采用非字段校验器配置风格时，该校验器的配置示例如下。

```
<validators>
    <!-- 使用非字段校验器配置风格来配置必填字符串校验器-->
    <validator type="requiredstring">
        <param name="fieldName">user.username</param>
        <param name="trim">true</param>
        <message>请输入用户名! </message>
    </validator>
```

```
</validators>
```

采用字段校验器配置风格时，该校验器的配置示例如下。

```
<validators>
    <!--使用字段校验器配置风格来配置必填字符串校验器-->
    <field name="user.username">
        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <message>请输入用户名</message>
        </field-validator>
    </field>
</validators>
```

3. 字符串长度验证器(stringlength validator)

StringLengthFieldValidator 验证器检查一个字符串字段值是否在一定的长度范围内。该验证器可以接受四个参数，分别如下。

- fieldname：指定要验证的字段名。如果使用<field>元素来声明字段验证器，则不需要这个参数。
- maxLength：指定字段值的最大长度。如果没有指定该参数，则不检查最大长度。
- minLength：指定字段值的最小长度。如果没有指定该参数，则不检查最小长度。
- trim：布尔值，指定在执行长度检测之前是否调用 String 的 trim()方法删除首尾的空格。默认值为 true。

采用非字段校验器配置风格时，该校验器的配置示例如下。

```
<validators>
    <!--使用非字段校验器配置风格来配置字符串长度校验器-->
    <validator type="stringlength">
        <param name="fieldName">user.username</param>
        <param name="minLength">10</param>
        <param name="maxLength">18</param>
        <param name="trim">true</param>
        <message>你输入的用户名长度必须在${minLength}到${maxLength}之间！
</message>
    </validator>
</validators>
```

采用字段校验器配置风格时，该校验器的配置示例如下。

```
<validators>
    <!--使用字段校验器配置风格来配置字符串长度校验器-->
    <field name="user.username">
        <field-validator type="stringlength">
            <param name="minLength">10</param>
            <param name="maxLength">18</param>
            <param name="trim">true</param>
            <message>你输入的用户名长度必须在${minLength}到${maxLength}之间！
</message>
```

```
        </field-validator>
    </field>
</validators>
```

4．整数验证器(int validator)

IntRangeFieldValidator 验证器检查指定的整数是否在一定的范围内。该验证器可以接受三个参数，如下。

- fieldname：指定要验证的字段名。如果使用<field>元素来声明该字段验证器，则不需要这个参数。
- min：指定整数的最小值。如果没有指定该参数，则不检查最小值。
- max：指定整数的最大值。如果没有指定该参数，则不检查最大值。

采用非字段校验器配置风格时，该校验器的配置示例如下。

```
<validators>
    <!--使用非字段校验器配置风格来配置整数校验器-->
    <validator type="int">
        <param name="fieldName">user.age</param>
        <param name="min">1</param>
        <param name="max">100</param>
        <message>你输入的年龄无效，必须在${min}到${max}之间！</message>
    </validator>
</validators>
```

采用字段校验器配置风格时，该校验器的配置示例如下。

```
<validators>
    <!--使用字段校验器配置风格来配置整数校验器-->
    <field name="user.age">
        <field-validator type="int">
            <param name="min">1</param>
            <param name="max">100</param>
            <message>你输入的年龄无效，必须在${min}到${max}之间！</message>
        </field-validator>
    </field>
</validators>
```

5．双精度浮点数验证器(double validator)

DoubleRangeFieldValidator 验证器检查指定的双精度浮点数是否在一定的范围内。该验证器可以接受五个参数，如下所示。

- fieldname：指定要验证的字段名。如果使用<field>元素来声明该字段验证器，则不需要这个参数。
- minInclusive：指定双精度浮点数的最小值，待检测的值可以等于这个最小值。如果没有指定该参数，则不检查这个值。
- maxInclusive：指定双精度浮点数的最大值，待检测的值可以等于这个最大值。如果没有指定该参数，则不检查这个值。

- minExclusive：指定双精度浮点数的最小值，待检测的值必须大于这个最小值。如果没有指定该参数，则不检查这个值。
- maxExclusive：指定双精度浮点数的最大值，待检测的值必须小于这个最大值。如果没有指定该参数，则不检查这个值。

采用非字段校验器配置风格时，该校验器的配置示例如下。

```
<validators>
    <!-- 使用非字段校验器配置风格来配置双精度浮点数验证器-->
    <validator type="double">
        <param name="fieldName">price</param>
        <param name="minInclusive">10.1</param>
        <param name="maxInclusive">100.1</param>
        <message>商品价格必须在${minInclusive}到${maxInclusive}范围内！
</message>
    </validator>
</validators>
```

采用字段校验器配置风格时，该校验器的配置示例如下。

```
<validators>
    <!-- 使用字段校验器配置风格来配置双精度浮点数验证器-->
    <field name="price">
        <field-validator type="double">
            <param name="minExclusive">10.123</param>
            <param name="maxExclusive">99.999</param>
            <message>商品价格必须在${minExclusive}到${maxExclusive}之间！
</message>
        </field-validator>
    </field>
</validators>
```

6. 日期验证器(date validator)

DateRangeFieldValidator 验证器检查给出的日期是否在指定的范围内。该验证器可以接受三个参数。

- fieldname：指定要验证的字段名。如果使用<field>元素来声明该字段验证器，则不需要这个参数。
- min：指定日期的最小值。如果没有指定该参数，则不检查最小值。
- max：指定日期的最大值。如果没有指定该参数，则不检查最大值。



如果没有指定日期转换器，框架将使用 XWorkBasicConverter 来进行日期转换，默认使用 Date.SHORT 格式来做日期转换，使用程序中指定的 locale 或者系统默认的 locale。

按照惯例，请读者来看看下面的代码。

```
<validators>
    <!-- 使用非字段校验器配置风格来配置日期校验器 -->
    <validator type="date">
```

```
        <param name="fieldName">user.birth</param>
        <param name="min">01/01/1990</param>
        <param name="max">01/01/2011</param>
        <message>出生日期必须在 1990 年 01 月 01 日到 2011 年 01 月 01 日之间!</message>
    </validator>

    <!-- 使用字段校验器配置风格来配置日期校验器 -->
    <field name="user.birth">
        <field-validator type="date">
            <param name="min">01/01/1990</param>
            <param name="max">01/01/2011</param>
            <message>出生日期必须在${min}到${max}之间！</message>
        </field-validator>
    </field>
</validators>
```

7. 表达式验证器(expression validator)

Expression Validator 是一个普通验证器(不能使用<field>元素来声明)，它基于 OGNL 表达式进行验证。该验证器可以接受一个参数：expression。这个参数指定要计算的 OGNL 表达式，该表达式基于值栈进行求值。表达式计算的结果必须是 Boolean 值，如果为 true，则验证通过；如果为 false，则验证失败。

采用非字段校验器配置风格时，该校验器的配置示例如下。

```
<validators>
    <validator type="expression">
        <param name="expression">user.password==verifyPassword</param>
        <message>两次输入的密码不一致！</message>
    </validator>
</validators>
```

8. 字段表达式验证器(fieldexpression validator)

FieldExpressionValidator 验证器使用 OGNL 表达式验证字段。该验证器可以接受两个参数，

- fieldname：指定要验证的字段名。如果使用<field>元素来声明该字段验证器，则不需要这个参数。
- expression：指定要计算的 OGNL 表达式，该表达式基于值栈进行求值。表达式计算的结果必须是 Boolean 值，如果为 true，则验证通过；如果为 false，则验证失败。

采用非字段校验器配置风格时，该校验器的配置示例如下。

```
<validators>
    <!-- 使用非字段校验器配置风格来配置字段表达式校验器-->
    <validator type="fieldexpression">
        <param name="fieldName">verifyPassword</param>
        <param name="expression">verifyPassword==user.password</param>
        <message>两次输入的密码不一致！</message>
    </validator>
</validators>
```

采用字段校验器配置风格时，该校验器的配置示例如下。

```
<validators>
    <!-- 使用字段校验器配置风格来配置字段表达式校验器-->
    <field name="verifyPassword">
        <field-validator type="fieldexpression">
            <param name="expression">verifyPassword==user.password</param>
            <message>两次输入的密码不一致！</message>
        </field-validator>
    </field>
</validators>
```

9. 正则表达式验证器(regex validator)

RegexFieldValidator 验证器使用正则表达式验证一个字符串字段值。该验证器可以接受四个参数。

- fieldname：指定要验证的字段名。如果使用<field>元素来声明该字段验证器，则不需要这个参数。
- expression：指定用于验证字段值的正则表达式。该参数是必需的。
- caseSensitive：布尔值，指定字段值是否按照大小写相关的方式来匹配正则表达式。该参数是可选的，默认值是 true。
- trim：布尔值，指定在进行正则表达式匹配之前，是否应该删除字符串首尾的空格。该参数是可选的，默认值是 true。

相信读者看完下面的代码会恍然大悟的。

```
<validators>
    <!-- 使用非字段校验器配置风格来配置正则表达式校验器-->
    <validator type="regex">
        <param name="fieldName">user.zipcode</param>
        <param name="expression"><![CDATA[[0-9]\d{5}(?!\d)]]></param>
        <message>邮政编码无效！</message>
    </validator>

    <!-- 使用字段校验器配置风格来配置正则表达式校验器 -->
    <field name="user.zipcode">
        <field-validator type="regex">
            <param name="expression"><![CDATA[[0-9]\d{5}(?!\d)]]></param>
            <message>邮政编码无效！</message>
        </field-validator>
    </field>
</validators>
```

10. 邮件地址验证器(email validator)

EmailValidator 验证器采用正则表达式验证一个字符串是否是合法的邮件地址，如果指定的字段值为 null 或者为“ ”，则该验证器只是简单的返回，并不将它报告为一个验证错误。

用于验证字符串是否合法邮件地址的正则表达式为：

```
\\b(^[_A-2a-z0-9-]+(\\.[_A-Za-z0-9-]+)*@([A-Za-z0-9-])+(\\.[A-Za-z0-9-]+)*(
(\\.[A-Za-z0-9]{2,})|(\\.[A-Za-z0-9]{2,}\\.[A-Za-z0-9]{2,}))$)\\b
```



随着技术的不断发展，上面的正则表达式可能不能完全覆盖实际的电子邮件地址。此时，建议开发者使用正则表达式校验器来完成邮件校验。

这个正则表达式已经内置在 EmailValidator 验证器中。EmailValidator 验证器可以接受一个参数：fieldName。这个参数指定要验证的字段名。如果使用<field>元素来声明该字段验证器，则不需要这个参数。

采用非字段校验器配置风格时，该校验器的配置示例如下。

```
<validators>
    <!-- 使用非字段校验器配置风格来配置邮件校验器-->
    <validator type="email">
        <param name="fieldName">user.email</param>
        <message>邮箱地址无效！</message>
    </validator>
</validators>
```

采用字段校验器配置风格时，该校验器的配置示例如下。

```
<validators>
    <!-- 使用字段校验器风格来配置邮件校验器，校验 user.email 属性-->
    <field name="user.email">
        <field-validator type="email">
            <message>邮箱地址无效！</message>
        </field-validator>
    </field>
</validators>
```

11. 网址验证器(url validator)

URLValidator 验证器检查指定的字段值是否是字符串，并且是合法的 URL。该验证器可以接受一个参数：fieldName。这个参数指定要验证的字段名。如果使用<field>元素来声明该字段验证器，则不需要这个参数。

下面的代码就是使用网址验证器的例子。

```
<validators>
    <!-- 使用非字段校验器风格来配置网址校验器-->
    <validator type="url">
        <param name="fieldName">homePage</param>
        <!--指定校验失败提示信息-->
        <message>无效的链接地址！</message>
    </validator>

    <!-- 使用字段校验器配置风格来配置网址校验器，校验 homePage 属性-->
    <field name="homePage">
        <field-validator type="url">
            <message>无效的链接地址！</message>
```

```
        </field-validator>
    </field>
</validators>
```

12．visitor验证器(visitor validator)

关于 visitor 验证器的用法，请参见后面章节。

13．转换验证器(conversion validator)

ConversionErrorFieldValidator 验证器检查指定字段在类型转换过程中是否出现转换错误。该验证器可以接受两个参数。

- fieldname：指定要验证的字段名。如果使用<field>元素来声明该字段验证器，则不需要这个参数。
- repopulateField：布尔值，指定出现类型转换错误时，是否保留字段的原始值。在出现类型转换错误时，请求会被导向到 INPUT 结果视图，当我们希望在发生错误的字段中显示原始输入的值时，应该将该参数设置为 true。

采用非字段校验器配置风格时，该校验器的配置示例如下。

```
<validators>
    <!-- 非字段校验器风格来配置转换校验器 -->
    <validator type="conversion">
        <param name="fieldName">intField</param>
        <!-- 指定类型转换失败后，返回输入页面保留原来的错误输入信息-->
        <param name="repopulateField">true</param>
        <message>不能转换成 Integer 类型！</message>
    </validator>
</validators>
```

采用字段校验器配置风格时，给校验器的配置示例如下。

```
<validators>
    <!-- 字段校验器配置风格来配置转换校验器-->
    <field name="intField">
        <field-validator type="conversion">
            <param name="repopulateField">true</param>
            <message>不能转换成 Integer 类型！</message>
        </field-validator>
    </field>
</validators>
```

6.3.2　实例描述

上次做的用户注册功能真是“失败”，我费了九牛二虎之力才做好。现在项目经理又让换……说到我们的项目经理这个人，我们几个都会皱紧眉头，他是一个性格有点古怪，脾气很倔的一个人，说一不二，管你有理没理，他说的你都得听。他看到我改写的用户注册输入验证规则文件后直接把项目给我发了过来，让我重写。问其原因，才知必须要用非字段校验器风格类配置所有的字段输入校验器。

6.3.3 实例应用

【例 6-3】 为用户注册输入数据校验改变“风格”。

修改校验规则文件(UserRegistAction-validation.xml)，使用非字段校验器风格来对用户输入的所有数据进行配置相应的校验器。修改后的内容如下。

```
<validators>
    <!-- 使用非字段校验器配置风格对 user.username 属性配置必填校验器 -->
    <validator type="requiredstring">
        <!-- 指定需要校验的字段名：user.username -->
        <param name="fieldName">user.username</param>
        <!-- 指定在执行长度检测之前删除首尾的空格 -->
        <param name="trim">true</param>
        <message key="error.username.required"/>
    </validator>
    <!-- 使用非字段校验器配置风格对 user.username 属性配置字符串长度校验器 -->
    <validator type="stringlength">
        <!-- 指定需要校验的字段名：user.username -->
        <param name="fieldName">user.username</param>
        <!-- 指定 user.username 属性字符串的最小长度 -->
        <param name="minLength">4</param>
        <!-- 指定 user.username 属性字符串的最大长度 -->
        <param name="maxLength">12</param>
        <!-- 指定校验失败的提示信息 -->
        <message key="error.username.length"/>
    </validator>
    <!-- 使用非字段校验器配置风格对 user.password 属性配置必填校验器 -->
    <validator type="requiredstring">
        <param name="fieldName">user.password</param>
        <!-- 指定校验失败后的提示信息 -->
        <message key="error.password.required"/>
    </validator>
    <!-- 使用非字段校验器配置风格对 user.password 属性配置字符串长度校验器 -->
    <validator type="stringlength">
        <param name="minLength">4</param>
        <param name="maxLength">8</param>
        <message key="error.password.length"/>
    </validator>
    <!-- 使用非字段校验器配置风格对 Action 中的 verifyPassword 属性配置必填校验器 -->
    <validator type="requiredstring">
        <!-- 指定要校验的字段：verifyPassword -->
        <param name="fieldName">verifyPassword</param>
        <message key="error.verifyPassword.required"/>
    </validator>
    <!-- 使用非字段校验器配置风格对 verifyPassword 属性配置字段表达式校验器 -->
    <validator type="fieldexpression">
        <param name="fieldName">verifyPassword</param>
```

```
        <!-- 指定逻辑表达式 -->
        <param name="expression">verifyPassword==user.password</param>
        <!-- 指定校验失败的提示信息 -->
        <message key="error.verifyPassword.identical"/>
    </validator>
    <!-- 使用非字段校验器对 user.age 属性配置整数校验器 -->
    <validator type="int">
        <param name="fieldName">user.age</param>
        <!-- 指定 user.age 属性的最小值 -->
        <param name="min">10</param>
        <!-- 指定 user.age 属性的最大值 -->
        <param name="max">100</param>
        <!-- 指定校验失败的提示信息 -->
        <message key="error.age.invalid"/>
    </validator>
    <!-- 使用非字段校验器风格对 user.email 属性配置必填校验器 -->
    <validator type="requiredstring">
        <param name="fieldName">user.email</param>
        <message key="error.email.required"/>
    </validator>
    <!-- 使用非字段校验器风格对 user.email 属性配置邮件地址校验器 -->
    <validator type="email">
        <param name="fieldName">user.email</param>
        <message key="error.email.invalid"/>
    </validator>
</validators>
```

6.3.4　运行结果

运行 userRegist.jsp 页面，输入信息不合法，验证失败，提示错误信息。如图 6-10 所示。

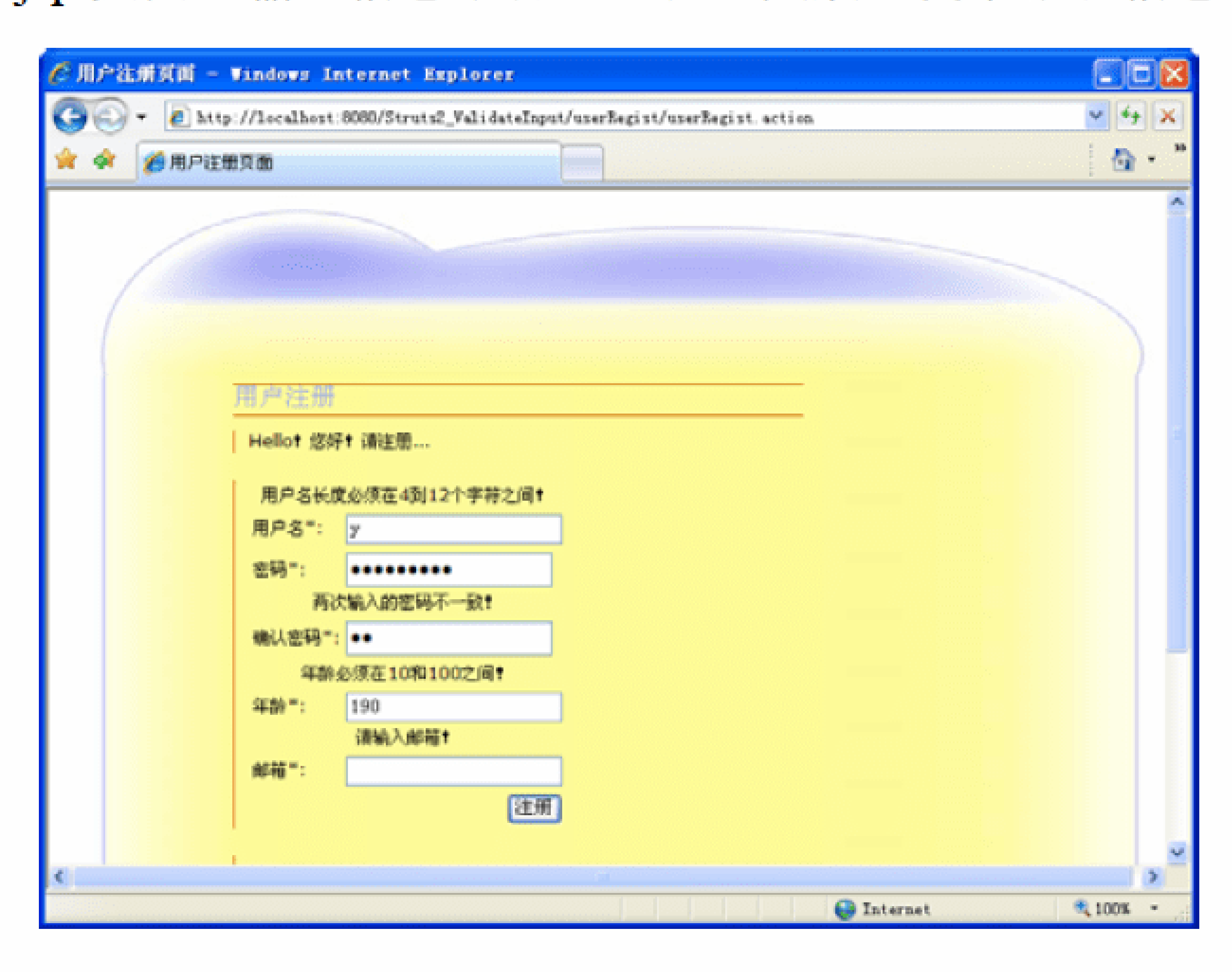


图 6-10　非字段校验器风格提示错误信息

6.3.5 实例分析



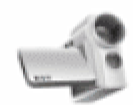
源码解析：

从上面的案例中可以得知校验器有两种风格的配置：非字段和字段校验器配置风格。在6.2.2 节我们使用了字段校验器配置风格配置了校验器，在本节中我们使用了非字段校验器配置风格，这两种配置风格都能完成对用户输入数据的校验工作，读者根据自己的爱好选择一种进行配置即可。

6.4 开发自己的验证器

看到这节的标题你一定心中充满了疑惑：“不是说 Struts 2 内置的验证器可以满足大部分的校验需求吗？怎么还要开发自己的验证器呢？”这和拦截器是一样的：有内置拦截器，也有自定义拦截器。也许正是因为 Struts 2 存在着这样的优势而成为目前 Web 开发中较为流行的框架。

下面就给读者介绍一下如何开发自己的验证器以及使用自定义验证器



视频教学：光盘/videos/06/MyValidate.avi　　　　长度：10 分钟

6.4.1 基础知识——开发属于自己的验证器

即使你对 Struts 2 内置验证程序的内部情况一无所知，也不影响你使用它们。可如果你想编写你自己的验证程序，就必须对用来实现 Struts 2 验证程序的各个类和它们的注册机制有一定的了解。

1. Validator接口介绍

验证程序必须实现 Validator 接口，它是 com.opensymphony.xwork2.validator 包的一部分。图 6-11 给出了这个接口、它的子接口和实现类。

在下图中，我省略了包的名字。Validator、FieldValidator 和 ShortCircuitableValidator 接口属于 com.opensymphony.xwork2.validator 包，其他的组件属于 com.opensymphony.xwork2.validator.validators 包。Validator 接口的定义如下。

```
package com.opensymphony.xwork2.validator;
public interface Validator{
    void setDefaultMessage(String message);
    String getDefaultMessage();
    String getMessage(Object object);
    void setMessageKey(String key);
    String getMessageKey();
    void setValidatorType(String type);
```

```
    String getValidatorType();
    void setValidatorContext(ValidatorContext validatorContext);
    ValidatorContext getValidatorContext();
    void validate(Object object) throws ValidationException;
}
```

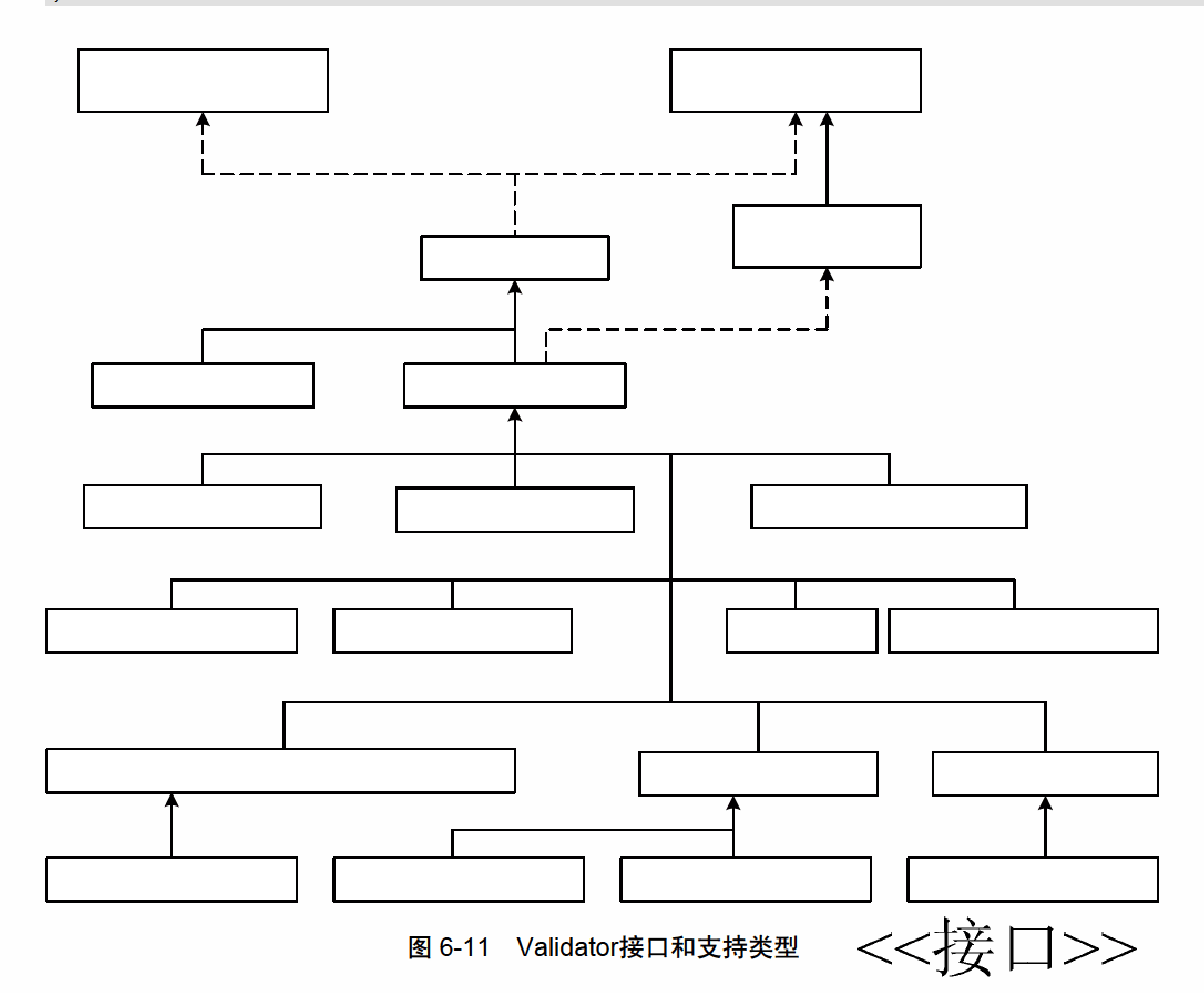


图 6-11　Validator接口和支持类型

Validation 拦截器负责加载执行各种验证程序。从上述源码中，可以知道在Validation 拦截器中加载了一个验证程序之后，这个拦截器将调用那个验证程序的 setValidatorContext 方法，并把当前的 ValidatorContext 对象传递给它，这样我们可以访问当前动作。接下来，Validation 拦截器将调用 validate 方法并把需要验证的对象传递给它。validate 方法是在编写一个自定义的验证程序时需要覆盖的方法。



对便捷类 ValidatorSupport 或 FieldValidatorSupport 进行扩展要比自行实现 Validator 接口容易的多。如果要创建一个普通的验证程序(非字段验证程序)，请扩展 ValidatorSupport 类；如果要编写一个字段验证程序，请扩展 FieldValidatorSupport 类；如果设计的验证程序能够接受一个输入参数，需要为这个参数增加一个相应的属性。例如，如果验证程序允许一个 minValue 参数，还需要增加一个名为 minValue 的属性，并为它编写 getter 方法和 setter 方法。

2. ValidatorSupport类介绍

从上图 6-11 可以看出，ValidatorSupport 类实现了 Validator 接口，在 ValidatorSupport 类中增加了几个方法，读者可以从验证程序类里调用它们。下面三个是便捷方法。

- protected java.lang.Object getFieldValue(java.lang.String name,java.lang.Object object) trows ValidationException：它返回 Object 对象的 name 字段的值。
- protected void addActionError(java.lang.Object actionError)：这个方法的主要功能是增加一个动作错误信息。
- protected void addFieldError(java.lang.String propertyName,java.lang.Object object)：这个方法的主要功能是增加一个字段错误信息。

如果读者编写的是一个非字段验证程序，在验证失败时需要从 validate 方法调用 addActionError 方法；如果读者编写的是一个字段验证程序，在验证失败时需要从 validate 方法调用 addFieldError 方法。

FieldValidatorSupport 类扩展了 ValidatorSupport 类并新增了 propertyType 和 fieldName 两个属性。

3. RequiredStringValidator类介绍

在对输入数据配置字符串非空验证校验器 requiredstring 时就是用 RequiredStringValidator 类实现的，它的源代码如下。

```
package com.opensymphony.xwork2.validator.validators;
import com.opensymphony.xwork2.validator.ValidationException;
public class RequiredStringValidator extends FieldValidatorSupport{
    private boolean doTrim=true;
    public void setTrim(boolean trim){
        doTrim=trim;
    }
    public boolean getTrim(){
        return doTrim;
    }
    public void validate(Object object) throws ValidationException{
        String fieldName=getFieldName();
        Object value=this.getFieldValue(fieldName,object);
        if(!(value instanceof String)){
            addFieldError(fieldName,object);
        }else{
            String s=(String)value;
            if(doTrim){
                s=s.trim();
            }
            if(s.length()==0){
                addFieldError(fieldName,object);
            }
        }
    }
}
```

从 RequiredStringValidator 类的源码中可以看出，requiredstring 验证程序可以接受一个 trim 参数，所以 RequiredStringValidator 类需要有一个相应的 trim 属性。如果有一个 trim 参数被传递给这个验证程序，Validation 拦截器就会调用 trim 属性的 setter 方法。validate 方法负责具体进行有关的验证。如果验证失败，这个方法必须调用 addFieldError 方法。

6.4.2　实例描述

前几天我 QQ 号被盗了，因为我的 QQ 密码被人给改了，后来我同事说 QQ 号一般不容易被盗的(要是容易被盗的话，腾讯早倒闭了！呵呵……)，但是我的为什么那么容易被人家猜中密码呢？究其原因，原来是因为我密码设的太简单。因为 QQ 号是公开的，只要密码猜中了，盗 QQ 就像是吃饭一样的容易。呵呵……我的密码是：123456789，所以很容易被别人猜中啊！所以我下定决心，再申请一个 QQ 号码，这次把密码设置的复杂点，安全性能强点的。至少要包括大、小写字母加数字吧！

同时我也想到了以后给客户做软件也不能疏忽这个问题，直接让用户注册账号时，输入的密码必须要复杂点，否则用户在本软件中的信息丢失了，损失可不小啊！

6.4.3　实例应用

【例 6-4】 为用户注册输入密码配置“强口令字”校验器。

这个例子的最终成果是一个用来检查口名字强度的 strongpass 验证程序：只有那些至少包含一个数字、一个小写字母和一个大写字母的口令才会被认为是一个强口令字。此外，这个验证程序还可以接受一个 minLength 参数，用户可以通过这个参数来设置一个可接受口令字的最小长度。

(1)　strongpass 验证程序的支持类是 com.struts2.validator.StrongPassValidator，这个类扩展自 FieldValidatorSupport 类。validate 方法将调用 isStrongPass(String fieldName)方法来测试口令字的强度。com.struts2.validator.StrongPassValidator 的内容如下。

```
package com.struts2.validator;
import com.opensymphony.xwork2.validator.ValidationException;
import com.opensymphony.xwork2.validator.validators.FieldValidatorSupport;
/**
 * 验证程序的支持类
 * @author Administrator
 *
 */
public class StrongPassValidator extends FieldValidatorSupport {
    private int minLength=-1;//定义文本的最小长度
    public int getMinLength() {
        return minLength;
    }
    public void setMinLength(int minLength) {
        this.minLength = minLength;
```

```
    }
    //验证程序
    public void validate(Object object) throws ValidationException {
        String fieldName=getFieldName();
        String value=(String)getFieldValue(fieldName,object);//获取用户输入数
据的值
        //如果用户输入的为 null 或""，验证失败，返回错误信息
        if(value==null||value.length()==0){
            return;
        }
        //如果定义的文本最小长度大于-1 且输入的数据值小于这里定义的最小长度，返回错误信息
        if((minLength>-1)&&(value.length()<minLength)){
            addFieldError(fieldName,object);//添加错误信息
        }
        else if(!isStrongPass(value)){//如果用户输入的数据不是强口令，则返回错误信息
            addFieldError(fieldName,object);
        }
    }
    //定义三个变量
    private static final String GROUP_1="abcdefghijklmnopqrstuvwxyz";
    private static final String GROUP_2="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    private static final String GROUP_3="0123456789";
    //是否是强口令
    protected boolean isStrongPass(String pass){
        //定义三个 boolean 变量，分别记录用户输入数据中是否有小写字母、大写字母、数字
        boolean con1=false;
        boolean con2=false;
        boolean con3=false;
        int length=pass.length();//获取输入的长度
        for(int i=0;i<length;i++){
            if(con1 && con2 && con3){//如果用户输入的数据中既有小写字母也有大写字母
                                      还有数字，则跳出循环
                break;
            }
            String character=pass.substring(i,i+1);//依次获取用户输入的数据字符
            if(GROUP_1.contains(character)){//如果检查的用户输入数据中存在小写字
                                             母，则设置 con1=true，继续遍历
                con1=true;
                continue;
            }
            if(GROUP_2.contains(character)){//如果检查出用户输入数据中存在大写字
                                             母，则设置 con2=true，继续遍历
                con2=true;
                continue;
            }
            if(GROUP_3.contains(character)){//如果检查出用户输入数据中存在数字，则
                                             设置 con3=true
                con3=true;
            }
```

```
        }
        return (con1 && con2 && con3);//返回用户输入数据中是否包含大、小写字母和数
                                        字，是一个 boolean 值
    }
}
```

(2) 在类路径下(src 目录下或 WEB-INF/classes 子目录下)新创建 validators.xml 文件，用于对 strongpass 验证程序进行注册。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC
    "-//OpenSymphony Group//XWork Validator Config 1.0//EN"
    "http://www.opensymphony.com/xwork/xwork-validator-config-1.0.dtd">
<validators>
    <!-- 配置 strongpass 校验器 -->
    <validator name="strongpass"
class="com.struts2.validator.StrongPassValidator"/>
</validators>
```

在注册了定制验证程序之后，就可以像使用 Struts 2 内置验证程序那样使用它了。

(3) 在这个例子中，还是用前面案例中的 User 类和 UserRegistAction 类，无需任何修改。需要修改 UserRegistAction-validation.xml 文件中的校验密码规则为以下所示。

```
    <!-- 使用非字段校验器配置风格对 user.password 属性配置必填校验器 -->
    <validator type="requiredstring">
        <param name="fieldName">user.password</param>
        <!-- 指定校验失败后的提示信息 -->
        <message key="error.password.required"/>
    </validator>
    <!-- 使用非字段校验器配置风格对 user.password 属性配置强命令校验器 -->
    <validator type="strongpass">
        <param name="fieldName">user.password</param>
        <param name="minLength">8</param>
        <message key="error.password.length"/>
    </validator>
```

(4) 修改国际化配置文件 UserRegistAction.properties 中的 error.password.length 值如下所示。

```
error.password.length=为了您的安全，密码必须包含大、小写字母和数字，且长度必须大于
${minLength}位!
```

6.4.4　运行结果

运行 userRegist.jsp 页面，当用户输入的密码没有同时包含数字、小写字母和大写字母，提示错误信息，如图 6-12 所示。

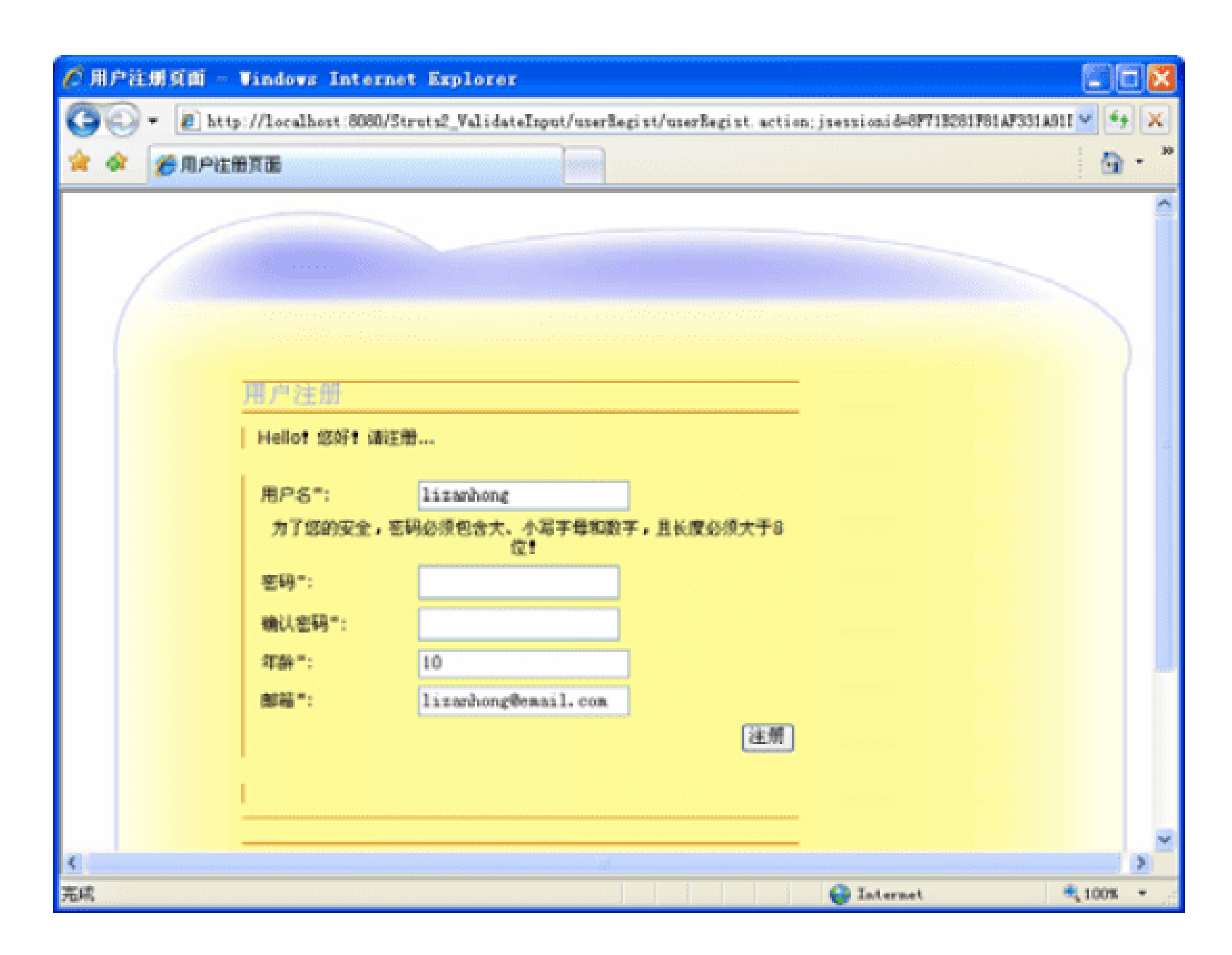
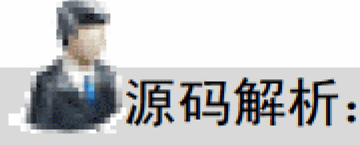


图 6-12　自定义校验器应用

6.4.5　实例分析



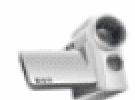
源码解析：

在上案例中，在类路径下(WEB-INF/classes 子目录下)的 validators.xml 文件中注册了一个名称为 strongpass 的校验器，使得在使用这个校验器时和使用 Struts 2 内置校验器完全一样，因此，可以在对密码配置强口令字校验器时，直接使用 type="strongpass"即可。同时在 StrongPassValidator 类中定义了变量 minLength，因此在 strongpass 校验器中配置 param name=“minLength”，在强口令字支持类(StrongPassValidator)中可获取 minlength 的值。

6.5　使用visitor字段验证器复用验证

前面一节我们是为 RegistAction 类编写了验证文件，但其中大多数的验证都是针对 User 对象的，如果有其他的 Action 也要使用 User 对象，就需要为这些 Action 重新编写针对 User 对象的验证规则。在实际应用中，经常会有这样的一种需求，就是为 domain 或 model 对象定义一些验证规则，然后在所有的 Action 或其他使用这些对象的类中复用它们。要做到这一点，可以利用 Struts 2 提供的 VisitorFieldValidator 验证器。

下面将介绍一下如何利用 visitor 验证程序。



视频教学：光盘/videos/06/visitor.avi　　长度：8 分钟

6.5.1　基础知识——VisitorFieldValidator验证器介绍

VisitorFieldValidator 验证器简称 visitor 验证器，它可以提高代码的可重用性，它告诉验证

框架查找 Action 的属性所属的类型对应的验证文件并进行验证。VisitorFieldValidator 可以处理简单的对象属性，也可以处理对象集合或者对象数组。

VisitorFieldValidator 验证器可以接受三个参数。

- fieldname：指定要验证的字段名。如果使用<field>元素来声明该字段验证器，则不需要这个参数。
- context：指定验证发生的上下文。该参数是可选的。
- appendPrefix：布尔值，指定要添加到字段上的前缀。该参数是可选的，默认值是 true。

无论做什么事都需要用事实来征服人，下面举个例子给读者介绍一下它的作用，相信读者会恍然大悟的！

假设有一个名为 Customer 的动作类，它有一个 Address 类型的 address 属性，而 Address 类又有 5 个属性(streetName、streetNumber、city、state 和 ZipCode)。为了验证给定的 Address 对象(它是 Customer 动作类的一个属性)的 ZipCode 属性，通常会在一个 Customer-validation.xml 文件里写出一个下面的 field 元素。

```
<field name="address.zipCode">
        <field-validator type="requiredstring">
            <message>Zip Code must not be empty</message>
        </field-validator>
</field>
```

忘记怎样才能在 OGNL 表达式里引用一个复杂对象了吗？

假设还有一个 Employee 动作类也使用 Address 作为一种属性类型。如果 Employee 的 address 属性需要使用与 Customer 中的 address 属性同样的验证规则，将有一个与 Customer-validation.xml 文件的内容完全一样的 Employee-validation.xml 文件。

这显然是一种冗余，而 visitor 验证程序可以帮助你把需要反复用到的验证规则提取出来单独保存为一个文件。以后，每当需要使用那些验证规则时，只需引用这个文件即可。在下面的例子里，针对 Address 类的验证规则将被单独保存到一个 Address-validation.xml 文件里，这使得 Customer-validation.xml 文件可以这样。

```
<field name="address">
        <!-- 使用 visitor 验证程序 -->
        <field-validator type="visitor">
            <message>Address:</message>
        </field-validator>
</field>
```

其中，这个 field 元素的含义是：address 属性将使用相关属性类型(Address)的验证配置文件来进行验证。换句话说，Struts 2 将使用 Address-validation.xml 文件来验证 address 属性。这样一来，即使有多个动作类都是用到了 Address，也不必在每个动作类的每个验证程序配置文件里写出同样的验证规则。

visitor(VisitorFieldValidator 验证器)验证程序的另一个功能是可以引用上下文。如果有多个使用了 Address 的动作，但其中有一个需要使用的验证规则与 Address-validation.xml 文件所给出的不一样，可以只为那个动作创建一个新的验证程序配置文件。这种验证程序配置文件的命名规则如下所示。

```
Address-context-validation.xml
```

其中的 context 是需要为 Address 类另行定义验证规则的动作的别名。例如，如果 AddEmployee 动作的 address 属性需要用与众不同的规则来进行验证，将需要创建这个文件。

```
Address-AddEmployee-validation.xml
```

到此还未结束。如果上下文的名字与动作别名不一样——不妨假设 AddManager 动作也需要使用 Address-AddEmployee-validation.xml 文件里的规则，而不是使用 Address- validation.xml 文件里的规则来进行验证。可以通过下面这样的 field 元素来告诉 visitor 验证程序需要使用另一种上下文。

```
<field name="address">
        <!-- 使用 visitor 验证程序 -->
        <field-validator type="visitor">
            <!--配置 context 参数-->
            <param name="context">specific</param>
            <message>Address:</message>
        </field-validator>
</field>
```

这向 visitor 验证程序表明：在验证 address 属性时，它应该使用 Address-specific-validation.xml 文件而不是 Address-AddManager-validation.xml 文件。

6.5.2 实例描述

最近项目组做了一个会员系统，即如果注册为我们公司网站的会员，可以享受购买商品七折的优惠。因为注册的用户越来越多，当然也有“滥竽充数”的人，填的信息是乱七八糟，让公司的“后勤”人员看的是“眼花缭乱”。于是项目组的人及时对这个会员系统进行维护，验证用户输入信息，如果填的不符合要求就不能成为公司的会员。

在做的过程中用到了一个 Address 类，这个类记录了用户的联系方式，而在已经做好的公司员工信息中已经用到了这个类，当然也用到了它的 Address-validation.xml 验证规则文件，不能有冗余的文件存在啊！怎么办？灵机一动，对，就用 VisitorFieldValidator 验证器来对注册会员进行联系方式验证……

6.5.3 实例应用

【例 6-5】 注册会员。

(1) 在 com.struts2.model 包下创建 Address 实体类，记载用户的一些联系方式信息，代码如下。

```
package com.struts2.model;
/**
*联系方式类
*@author Administrator
```

```
*
*/
public class Address {
    private String streetName;//街道名称
    private String streetNumber;//门牌号
    private String city;//城市
    private String state;//国家
    private String zipCode;//邮政编码
    /*下面是上面所有属性的 get、set 方法，这里省略*/
}
```

(2) 继续在 com.struts2.model 包下创建 Address 类的验证规则文件 Address-validation.xml。对 Address 类中所有的属性配置相关的校验器，对其进行输入校验，验证内容如下。

```
<validators>
    <!-- 为 Address 类中的 streetName 属性配置必填校验器 -->
    <field name="streetName">
        <field-validator type="requiredstring">
            <message>街道名称不能是空的！</message>
        </field-validator>
    </field>
    <!-- 为 Address 类中的 streetNumber 属性配置必填校验器 -->
    <field name="streetNumber">
        <field-validator type="requiredstring">
            <message>街道号码不能为空！</message>
        </field-validator>
    </field>
    <!-- 为 Address 类中的 city 属性配置必填校验器 -->
    <field name="city">
        <field-validator type="requiredstring">
            <message>城市不能为空！</message>
        </field-validator>
    </field>
    <!-- 为 Address 类中的 state 属性配置必填校验器 -->
    <field name="state">
        <field-validator type="requiredstring">
            <message>国家不能为空！</message>
        </field-validator>
    </field>
    <!-- 为 Address 类中的 zipCode 属性配置必填校验器 -->
    <field name="zipCode">
        <field-validator type="requiredstring">
            <message>邮政编码不能为空！</message>
        </field-validator>
    </field>
</validators>
```

(3) 在 com.struts2.actions 包下创建 Customer 类，继承自 com.opensymphony.xwork2.ActionSupport，并重写父类的 execute()方法，转入成功页面。代码如下。

```
package com.struts2.actions;
```

```
import com.opensymphony.xwork2.ActionSupport;
import com.struts2.model.Address;
/**
*客户 Action
*@author Administrator
*
*/
public class Customer extends ActionSupport {
    private String name;
    private Address address;
    @Override
    public String execute() throws Exception {
        return SUCCESS;
    }
    /*下面是上面 name、Address 对象 address 属性的 get、set 方法，这里省略*/
}
```

(4) 继续在 com.struts2.actions 包下新创建 Customer 类的验证规则文件 Customer-validation.xml，对 Customer 的属性 name、Address 对象 address 进行输入校验。校验内容如下。

```
<validators>
    <field name="name">
        <!-- 为 Customer 类中的 name 属性配置必填校验器 -->
        <field-validator type="requiredstring">
            <message>姓名不能为空！</message>
        </field-validator>
    </field>
    <!--使用 visitor 验证程序 -->
    <field name="address">
        <field-validator type="visitor">
            <!--为 VisitorFieldValidator 校验器配置 context 参数，指定在验证 address
属性时，它应该使用 Address-specific-validation.xml 文件-->
            <param name="context">specific</param>
            <message>Address:</message>
        </field-validator>
    </field>
</validators>
```

(5) 对邮政编码的验证除了非空之外还必须是有效的格式，而在 Address-validation.xml 文件中只进行了非空验证，并没有进行格式验证，因此需要在 Address 类所在的包 com.struts2.model 下创建 Address-specific-validation.xml 校验规则文件，对用户输入的邮政编码进行表达式验证。表达式校验器内容如下。

```
<validators>
    <!-- 为 Address 类中的 zipCode 属性配置表达式校验器 -->
    <field name="zipCode">
        <field-validator type="regex">
            <param name="expression">
                <![CDATA[\d\d\d\d\d\d]]>
```

```
            </param>
            <message>邮政编号必须有效！</message>
        </field-validator>
    </field>
</validators>
```

(6) 在 struts-config 文件夹下创建 customer.xml 文件，配置 Customer 类。配置如下。

```
    <package name ="customer" extends ="struts-default" >
        <!-- 配置一个名为 customer 的 Action-->
        <action name ="customer" class ="com.struts2.actions.Customer" >
            <result name="input">/customer.jsp</result>
            <result name="success">/success.jsp</result>
        </action>
    </package>
```

(7) 把 customer.xml 文件引入 struts.xml 文件当中。
(8) 创建会员注册页面 customer.jsp，代码如下。

```
<s:form action="customer.action" method="post">
            <s:textfield name="name" label="姓名" />
            <s:textfield name="address.streetName" label="街道名称"/>
            <s:textfield name="address.streetNumber" label="门牌号" />
            <s:textfield name="address.city" label="城市"/>
            <s:textfield name="address.state" label="国家"/>
            <s:textfield name="address.zipCode" label="邮政编码"/>
            <s:submit value="确定"/>
</s:form>
```

6.5.4　运行结果

运行 customer.jsp 页面，单击“确定”按钮，提交表单，提示错误信息，如图 6-13 所示。

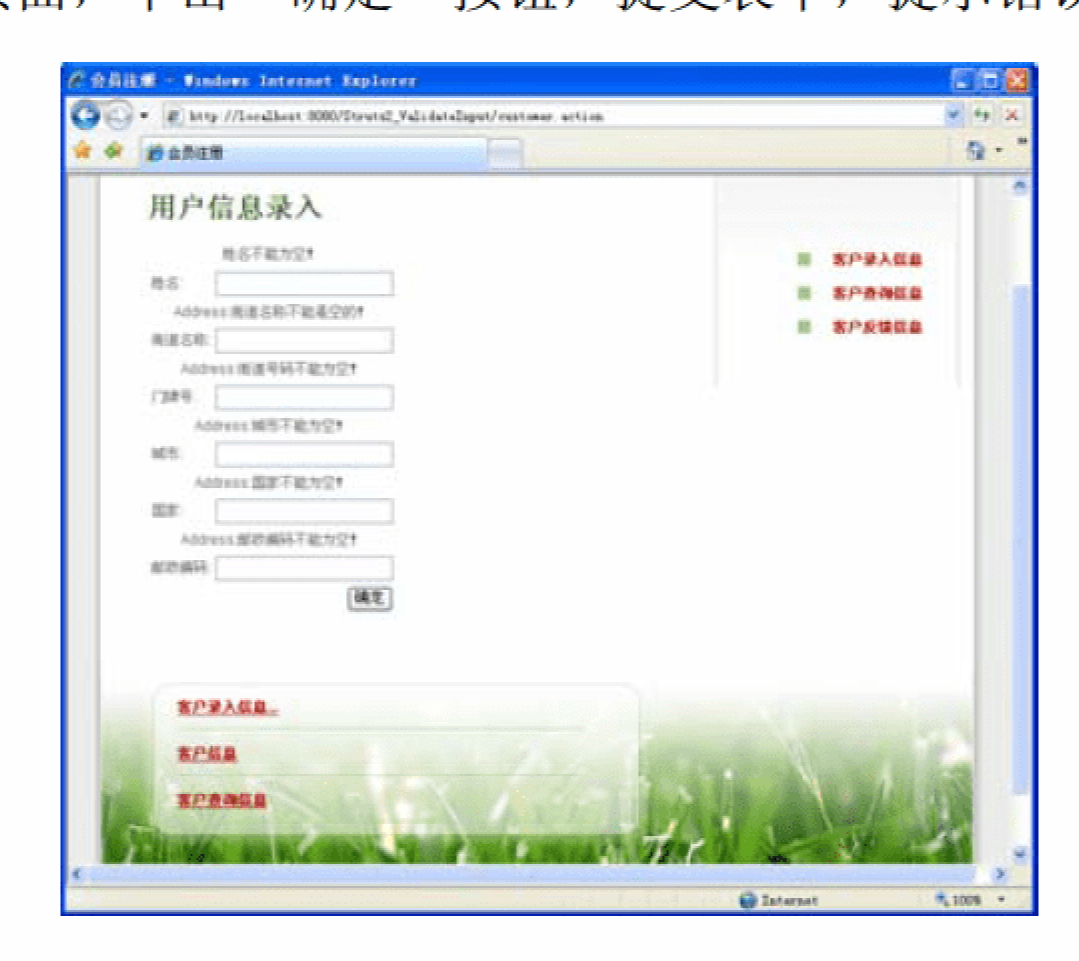


图 6-13　输入为空时提示错误信息

当输入所有信息，但邮政编码不为 6 位数字时，提示“邮政编码必须有效！”的错误信息，如图 6-14 所示。



图 6-14　邮政编码无效时提示错误信息

6.5.5　实例分析



源码解析：

在上案例中，Customer 动作类中有一个 Address 类型的 address 属性，而这个 Address 类所在的包为 com.struts2.model，因此需要在 com.struts2.model 包下创建 Address 类的校验规则文件 Address-validation.xml。

在对 Customer 类中的属性 address 进行校验时，使用了 visitor 验证程序，并配置了它的 context 参数值为 specific，这就说明在验证 address 属性时，不仅要使用 Address 类的校验规则文件 Address-validation.xml，还需要使用 Address-specific-validation.xml 文件。因为使用 visitor 验证程序时，是对 Customer 类中的属性 address 进行校验，因此 Address-specific-validation.xml 文件也需要放置在 com.struts2.model 包下。

6.6　使用验证注解

除了提供编写验证文件之外，Struts 2 还提供了使用注解的方式来定义验证规则。下面就来介绍一下可以使用哪些注解来对输入数据进行校验。



视频教学：光盘/videos/06/required.avi　　长度：8 分钟

6.6.1　基础知识——使用验证注解

Struts 2 内置的 13 个验证器都有对应的注解，下面来看一下 Struts 2 中与验证相关的注解

都有哪些。

1. RequiredFieldValidator注解

RequiredFieldValidator 注解对应于 RequiredFieldValidator 验证器，该注解只能用在方法级别，它们的参数如表 6-1 所示。

表 6-1　RequiredFieldValidator注解的参数

参　数	类　型	是否必需	默 认 值	描　述
message	String	是	无	验证失败时的默认错误消息
key	String	否	无	国际化资源文件中的消息 key
fieldName	String	否	无	对于 SIMPLE 验证类型，指定字段名
shortCircuit	boolean	否	false	验证器是否作为短路使用
type	ValidatorType	否	ValidatorType.FIELD	验证类型，可选的值为 FIELD 或 SIMPLE



message 参数用于配置默认的错误消息，如果使用了 key 参数，那么将以国际化资源文件中配置的消息文本作为错误消息；如果在资源文件中没有找到 key 对应的消息文本，则使用 message 参数给出的错误消息。message 参数也可以单独使用。

配置例子如下。

```
@RequiredFieldValidator(
            message="用户名不能为空！",
            key="error.username.required",
            shortCircuit=true
)
```

2. RequiredStringValidator注解

RequiredStringValidator 注解对应于 RequiredStringValidator 验证器，该注解只能用在方法级别，它的参数如表 6-2 所示。

表 6-2　RequiredStringValidator注解的参数

参　数	类　型	是否必需	默 认 值	描　述
message	String	是	无	验证失败时的默认错误消息
key	String	否	无	国际化资源文件中的消息 key
fieldName	String	否	无	对于 SIMPLE 验证类型，指定字段名
shortCircuit	boolean	否	false	验证器是否作为短路使用
type	ValidatorType	否	ValidatorType.FIELD	验证类型，可选的值为 FIELD 或 SIMPLE
trim	boolean	否	true	指定在进行是否为空检查之前，是否要删除字符串首尾的空格

配置例子如下。

```
@RequiredStringValidator(
            message="邮箱地址不能为空！",
            key="error.email.required",
            shortCircuit=true,
            trim=true
)
```

3. StringLengthFieldValidator注解

StringLengthFieldValidator 注解对应于 StringLengthFieldValidator 验证器，该注解只能用在方法级别，它的参数如表 6-3 所示。

表 6-3　StringLengthFieldValidator注解的参数

参　数	类　型	是否必需	默 认 值	描　述
message	String	是	无	验证失败时的默认错误消息
key	String	否	无	国际化资源文件中的消息key
fieldName	String	否	无	对于 SIMPLE 验证类型，指定字段名
shortCircuit	boolean	否	false	验证器是否作为短路使用
type	ValidatorType	否	ValidatorType.FIELD	验证类型，可选的值为 FIELD 或 SIMPLE
trim	boolean	否	true	指定在进行是否为空检查之前，是否要删除字符串首尾的空格
minLength	String	否	无	指定字符串的最小长度
maxLength	String	否	无	指定字符串的最大长度

配置例子如下。

```
@StringLengthFieldValidator(
            message="用户名必须在${minLength}和${maxLength}之间！",
            key="error.username.length",
            shortCircuit=true,
            trim=true,
            minLength="4",
            maxLength="12"
)
```

4. IntRangeFieldValidator注解

IntRangeFieldValidator 注解对应于 IntRangeFieldValidator 验证器，该注解只能用在方法级别，它的参数如表 6-4 所示。

表 6-4　IntRangeFieldValidator注解的参数

参　数	类　型	是否必需	默 认 值	描　述
message	String	是	无	验证失败时的默认错误消息
key	String	否	无	国际化资源文件中的消息 key
fieldName	String	否	无	对于 SIMPLE 验证类型，指定字段名
shortCircuit	boolean	否	false	验证器是否作为短路使用
type	ValidatorType	否	ValidatorType.FIELD	验证类型，可选的值为 FIELD 或 SIMPLE
min	String	否	无	指定整数的最小值
max	String	否	无	指定整数的最大值

配置例子如下。

```
@IntRangeFieldValidator(
            message="年龄必须在${min}和${max}之间！",
            key="error.age",
            shortCircuit=true,
            min="20",
            max="50"
)
```

5. DoubleRangeFieldValidator注解

DoubleRangeFieldValidator 注解对应于 DoubleRangeFieldValidator 验证器，该注解只能用在方法级别，它的参数如表 6-5 所示。

表 6-5　DoubleRangeFieldValidator注解的参数

参　数	类　型	是否必需	默 认 值	描　述
message	String	是	无	验证失败时的默认错误消息
key	String	否	无	国际化资源文件中的消息 key
fieldName	String	否	无	对于 SIMPLE 验证类型，指定字段名
shortCircuit	boolean	否	false	验证器是否作为短路使用
type	ValidatorType	否	ValidatorType.FIELD	验证类型，可选的值为 FIELD 或 SIMPLE
minInclusive	String	否	无	指定双精度浮点数可包含的最小值
maxInclusive	String	否	无	指定双精度浮点数可包含的最大值
minExclusive	String	否	无	指定双精度浮点数必须大于的最小值
maxExclusive	String	否	无	指定双精度浮点数必须小于的最大值

如果没有指定最小值和最大值，那么 DoubleRangeFieldValidator 验证器将什么也不做。配置例子如下。

```
@DoubleRangeFieldValidator(
            message="价格必须在${minInclusive}和${maxInclusive}之间！",
            key="error.price",
            shortCircuit=true,
            minInclusive="20.1",
            maxInclusive="50.1"
)
```

6. DateRangeFieldValidator注解

DateRangeFieldValidator 注解对应于 DateRangeFieldValidator 验证器，该注解只能用在方法级别，它的参数如表 6-6 所示。

表 6-6　DateRangeFieldValidator注解的参数

参　数	类　型	是否必需	默 认 值	描　述
message	String	是	无	验证失败时的默认错误消息
key	String	否	无	国际化资源文件中的消息key
fieldName	String	否	无	对于 SIMPLE 验证类型，指定字段名
shortCircuit	boolean	否	false	验证器是否作为短路使用
type	ValidatorType	否	ValidatorType.FIELD	验证类型，可选的值为 FIELD 或 SIMPLE
minInclusive	String	否	无	指定双精度浮点数可包含的最小值
maxInclusive	String	否	无	指定双精度浮点数可包含的最大值
minExclusive	String	否	无	指定双精度浮点数必须大于的最小值
maxExclusive	String	否	无	指定双精度浮点数必须小于的最大值

配置例子如下。

```
@DateRangeFieldValidator(
            message="出生日期必须在${min}和${max}之间！",
            key="error.birthday",
            shortCircuit=true,
            min="1990/01/01",
            max="2011/01/01"
)
```

7. ExpressionValidator注解

ExpressionValidator 注解对应于 ExpressionValidator 验证器，该注解只能用在方法级别，它的参数如表 6-7 所示。

表 6-7　ExpressionValidator注解的参数

参　数	类　型	是否必需	默 认 值	描　述
message	String	是	无	验证失败时的默认错误消息
key	String	否	无	国际化资源文件中的消息 key
shortCircuit	boolean	否	false	验证器是否作为短路使用
expression	String	是	无	指定一个返回布尔值的 OGNL 表达式

配置例子如下。

```
@ExpressionValidator(
            message="两次输入的密码必须一致！",
            key="error.verifyPassword.identical",
            shortCircuit=true,
            expression="user.password==verifyPassword"
)
```

8. FieldExpressionValidator注解

FieldExpressionValidator 注解对应于 FieldExpressionValidator 验证器，该注解只能用在方法级别，它的参数如表 6-8 所示。

表 6-8　FieldExpressionValidator注解的参数

参　数	类　型	是否必需	默 认 值	描　述
message	String	是	无	验证失败时的默认错误消息
key	String	否	无	国际化资源文件中的消息 key
fieldName	String	否	无	对于 SIMPLE 验证类型，指定字段名
shortCircuit	boolean	否	false	验证器是否作为短路使用
expression	String	是	无	指定一个返回布尔值的 OGNL 表达式

配置例子如下。

```
@FieldExpressionValidator(
            fieldName="verifyPassword",
            expression="user.password.equals(verifyPassword)",
            key="error.verifyPassword.identical",
            message="两次输入的密码必须相同！"
)
```

9. RegexFieldValidator注解

RegexFieldValidator 注解对应于 RegexFieldValidator 验证器，该注解只能用在方法级别，

它的参数如表 6-9 所示。

表 6-9 RegexFieldValidator注解的参数

参 数	类 型	是否必需	默 认 值	描 述
message	String	是	无	验证失败时的默认错误消息
key	String	否	无	国际化资源文件中的消息 key
fieldName	String	否	无	对于 SIMPLE 验证类型，指定字段名
shortCircuit	boolean	否	false	验证器是否作为短路使用
type	ValidatorType	否	ValidatorType.FIELD	验证类型，可选的值为 FIELD 或 SIMPLE
expression	String	是	无	用于验证字段值的正则表达式

配置例子如下。

```
@RegexFieldValidator(
            message="不是有效的邮政编码！",
            key="regex.field",
            expression="<![CDATA[[0-9]\d{5}(?!\d)]]>"
)
```

10. EmailValidator注解

EmailValidator 注解对应于 EmailValidator 验证器，该注解只能用在方法级别，它的参数如表 6-10 所示。

表 6-10 EmailValidator注解的参数

参 数	类 型	是否必需	默 认 值	描 述
message	String	是	无	验证失败时的默认错误消息
key	String	否	无	国际化资源文件中的消息 key
fieldName	String	否	无	对于 SIMPLE 验证类型，指定字段名
shortCircuit	boolean	否	false	验证器是否作为短路使用
type	ValidatorType	否	ValidatorType.FIELD	验证类型，可选的值为 FIELD 或 SIMPLE

配置例子如下。

```
@EmailValidator(
            key="error.email.invalid",
            type=ValidatorType.FIELD,
            message="邮箱地址必须是有效的！"
)
```

11. UrlValidator注解

UrlValidator 注解对应于 UrlValidator 验证器，该注解只能用在方法级别，它的参数如表 6-11 所示。

表 6-11　UrlValidator注解的参数

参　数	类　型	是否必需	默 认 值	描　述
message	String	是	无	验证失败时的默认错误消息
key	String	否	无	国际化资源文件中的消息 key
fieldName	String	否	无	对于 SIMPLE 验证类型，指定字段名
shortCircuit	boolean	否	false	验证器是否作为短路使用
type	ValidatorType	否	ValidatorType.FIELD	验证类型，可选的值为 FIELD 或 SIMPLE

配置例子如下。

```
@UrlValidator(
            message="链接地址必须是有效的！",
            key="error.url",
            shortCircuit=true
)
```

12. VisitorFieldValidator注解

VisitorFieldValidator 注解对应于 VisitorFieldValidator 验证器，该注解只能用在方法即被，它的参数如表 6-12 所示。

表 6-12　VisitorFieldValidator注解的参数

参　数	类　型	是否必需	默 认 值	描　述
message	String	是	无	验证失败时的默认错误消息
key	String	否	无	国际化资源文件中的消息 key
fieldName	String	否	无	对于 SIMPLE 验证类型，指定字段名
shortCircuit	boolean	否	false	验证器是否作为短路使用
context	String	否	无	指定验证发生的上下文
appendPrefix	boolean	否	true	指定要添加到字段上的前缀

配置例子如下。

```
@VisitorFieldValidator(
            message="User:",
            shortCircuit=true,
            context="login",
            appendPrefix=true
)
```

13. ConversionErrorFieldValidator注解

ConversionErrorFieldValidator 注解对应于 ConversionErrorFieldValidator 验证器，该注解只能用在方法级别，它的参数如表 6-13 所示。

表 6-13　ConversionErrorFieldValidator注解的参数

参　数	类　型	是否必需	默 认 值	描　述
message	String	是	无	验证失败时的默认错误消息
key	String	否	无	国际化资源文件中的消息 key
fieldName	String	否	无	对于 SIMPLE 验证类型，指定字段名
shortCircuit	boolean	否	false	验证器是否作为短路使用
type	ValidatorType	否	ValidatorType.FIELD	验证类型，可选的值为 FIELD 或 SIMPLE

配置例子如下。

```
@ConversionErrorFieldValidator(
            message="类型转换失败！",
            key="i18n.key",
            shortCircuit=true
)
```

14. CustomValidator注解

CustomValidator 注解用于自定义验证器，通过 ValidationParameter 注解向自定义验证器提供参数。CustomValidator 注解可以用在方法或者类型级别，它的参数如表 6-14 所示。

表 6-14　CustomValidator注解的参数

参　数	类　型	是否必需	默 认 值	描　述
type	String	是	无	指定已注册的验证器的名字
message	String	是	无	验证失败时的默认错误消息
key	String	否	无	国际化资源文件中的消息 key
fieldName	String	否	无	对于 SIMPLE 验证类型，指定字段名
shortCircuit	boolean	否	false	验证器是否作为短路使用
parameters	ValidationParameter[]	否	无	为自定义验证器提供参数

配置例子如下。

```
@CustomValidator(
            type="validationCodeValidator",
            fieldName="validationCode",
            key="error.validationCode.invalid",
            message="验证字段必须是有效的！",
            parameters=
            {
                    @ValidationParameter(
```

```
                    name="sessionValidationCode",
                    value="#session.validationCode"
                )
            }
)
```

15．ValidationParameter注解

ValidationParameter 注解用于为自定义验证器提供参数，该注解必须在 CustomValidator 注解内部使用，作为 parameters 参数的元素值使用。ValidationParameter 注解的参数如表 6-15 所示。

表 6-15　ValidationParameter注解的参数

参　数	类　型	是否必需	默 认 值	描　述
name	String	是	无	参数的名字
value	String	是	无	参数的值

16．Validations注解

如果想要在一个方法上使用相同类型的多个注解，那么必须将它们放到@Validations()注解的内部使用。Validations 注解只能用在方法级别，它的参数如表 6-16 所示。

表 6-16　Validations注解的参数

参　数	类　型	是否必需	默 认 值	描　述
requiredFields	RequiredFieldValidator[]	否	无	RequiredFieldValidator 注解的数组
customValidators	CustomValidator[]	否	无	CustomValidator 注解的数组
conversionErrorFields	ConversionErrorFieldValidator[]	否	无	ConversionErrorFieldValidator 注解的数组
dateRangeFields	DateRangeFieldValidator[]	否	无	DateRangeFieldValidator 注解的数组
emails	EmailValidator[]	否	无	EmailValidator 注解的数组
fieldExpression	FieldExpressionValidator[]	否	无	EmailValidator 注解的数组
intRangeFields	IntRangeFieldValidator[]	否	无	IntRangeFieldValidator 注解的数组
requiredStrings	RequiredStringValidator[]	否	无	RequiredStringValidator 注解的数组
stringLengthFields	StringLengthFieldValidator[]	否	无	StringLengthFieldValidator 注解的数组
urls	UrlValidator[]	否	无	UrlValidator 注解的数组
visitorFields	VisitorFieldValidator[]	否	无	VisitorFieldValidator 注解的数组
regexFields	RegexFieldValidator[]	否	无	RegexFieldValidator 注解的数组
expression	ExpressionValidator[]	否	无	ExpressionValidator 注解的数组

配置例子如下。

```
@Validations(
        requiredStrings=
        {
                @RequiredStringValidator(
                        type=ValidatorType.SIMPLE,
                        fieldName="verifyPassword",
                        message="确认密码不能为空！",
                        key="error.verifyPassword.required"
                ),
                @RequiredStringValidator(
                        type=ValidatorType.SIMPLE,
                        fieldName="validationCode",
                        key="error.validationCode.required",
                        message="验证字段不能为空！"
                )

        },
        fieldExpressions=
        {
                @FieldExpressionValidator(
                        fieldName="verifyPassword",

expression="user.password.equals(verifyPassword)",
                        key="error.verifyPassword.identical",
                        message="两次输入的密码必须一致！"
                )
        },
        customValidators=
        {
                @CustomValidator(
                        type="validationCodeValidator",
                        fieldName="validationCode",
                        key="error.validationCode.invalid",
                        message="验证字段必须是有效的！",
                        parameters=
                        {
                                @ValidationParameter(
                                        name="sessionValidationCode",
                                        value="#session.validation Code"
                                )
                        }
                )
        }
)
```

17. Validation注解

使用 Validation 注解，可以在类或者接口上配置验证规则。Validation 注解只能用在类型级别，它的参数如表 6-17 所示。

表 6-17　Validation注解的参数

参　数	类　型	是否必需	默 认 值	描　述
validations	Validations[]	否	无	用于类或接口的验证规则

配置例子如下。

```
@Validations(
        requiredStrings=
        {
            @RequiredStringValidator(
                    type=ValidatorType.SIMPLE,
                    fieldName="verifyPassword",
                    message="请再次输入密码！",
                    key="error.verifyPassword.required"
            ),
            @RequiredStringValidator(
                    type=ValidatorType.SIMPLE,
                    fieldName="validationCode",
                    key="error.validationCode.required",
                    message="验证字段不能是空的！"
            )

    },
    fieldExpressions=
    {
            @FieldExpressionValidator(
                    fieldName="verifyPassword",
                    expression="user.password.equals(verifyPassword)",
                    key="error.verifyPassword.identical",
                    message="两次输入的密码必须一致！"
            )
    },
    customValidators=
    {
            @CustomValidator(
                    type="validationCodeValidator",
                    fieldName="validationCode",
                    key="error.validationCode.invalid",
                    message="验证字段必须是有效的！",
                    parameters=
                    {
                            @ValidationParameter(
                                    name="sessionValidationCode",
                                    value="#session.validation Code"
                            )
                    }
            )
        }
)
```



Struts 2 的文档对 Validation 注解的说明是："如果想要使用基于注解的验证，必须使用 Validation 注解标注类或者接口"，但经过测试，即使不使用 Validation 注解，只使用其他注解配置的验证规则也能正常地执行。

6.6.2 实例描述

最近一段时间由于公司业务繁忙，所以需要招新人。今天来了一个应聘的人，我接待了他。口头面试很成功，接下来的是笔试，我想着：这下得来点有难度的题来考考他，免得让他觉得我们公司实力不行。于是，我想到了之前为客户做的用户注册功能，让他用注解的形式来对用户输入数据进行校验，看他能力到底是强，还是弱。

6.6.3 实例应用

【例 6-6】 使用验证注解完成用户注册功能。

(1) 在 com.struts2.model 包下创建 Register.properties 文件，用于配置国际化信息。当用户输入错误时，读取此文件中的 key 值，获取提示的错误信息。内容如下。

```
error.username.required=您必须输入用户名!
error.username.length=您输入的用户名长度必须在${minLength}到 2${maxLength}之间!
error.password.required=您必须输入密码!
error.password.length=您输入的密码长度必须在${minLength}到${maxLength}之间!
error.email.required=你必须输入邮箱地址!
error.email.invalid=邮箱地址无效!
```

(2) 继续在 com.struts2.model 包下创建 Register 实体类，使用验证注解定义验证规则。内容如下。

```
package com.struts2.model;
import com.opensymphony.xwork2.validator.annotations.EmailValidator;
import
com.opensymphony.xwork2.validator.annotations.RequiredStringValidator;
import
com.opensymphony.xwork2.validator.annotations.StringLengthFieldValidator;
import com.opensymphony.xwork2.validator.annotations.ValidatorType;
public class Register{
    private String username;//用户名
    private String password;//密码
    private String email;//邮箱
    private String phone;//电话
    private String sex;//性别
    public String getUsername() {
        return username;
    }
    /**
```

```
 * 为 username 属性配置必填信息校验器
 */
@RequiredStringValidator(
        type=ValidatorType.FIELD,
        key="error.username.required",
        //添加短路校验器，否则如果用户名没有输入，会提示所有对 username 属性校验失败
          的错误信息
        shortCircuit=true
)
/**
 * 为 username 属性配置字符串长度校验器
 */
@StringLengthFieldValidator(
        type=ValidatorType.FIELD,
        key="error.username.length",
        minLength="4",
        maxLength="12"
)
public void setUsername(String username) {
    this.username = username;
}
public String getPassword() {
    return password;
}
/**
 * 为 password 属性配置必填信息校验器
 */
@RequiredStringValidator(
        type=ValidatorType.FIELD,
        key="error.password.required",
        //添加短路校验器
        shortCircuit=true
)
/**
 * 为 password 属性配置字符串长度校验器
 */
@StringLengthFieldValidator(
        type=ValidatorType.FIELD,
        key="error.password.length",
        minLength="4",
        maxLength="8"
)
public void setPassword(String password) {
    this.password = password;
}
public String getEmail() {
    return email;
}
/**
```

```
     * 为邮箱地址配置必填信息校验器
     * @param email
     */
    @RequiredStringValidator(
            type=ValidatorType.FIELD,
            key="error.email.required",
            //添加短路校验器
            shortCircuit=true
    )
    /**
     * 为邮箱地址配置邮箱格式校验器
     */
    @EmailValidator(
            key="error.email.invalid",
            type=ValidatorType.FIELD
    )
    public void setEmail(String email) {
        this.email = email;
    }
    /*下面是上面 phone、sex 两个属性的 set、get 方法，这里省略*/
}
```



在这里要提醒读者的是：验证注解在 getter 或者 setter 方法上使用都可以。

在上面的代码中，提到了一个新的知识点——短路校验器。短路校验器非常有用，读者可以把上面配置的 shortCircuit=true 去掉试试，当在用户名输入框中没有输入任何内容时，会提示两条错误信息，分别是“您必须输入用户名！”和“您输入的用户名长度必须在${minLength}到 2${maxLength}之间！”，这样是不是很不友好啊！

对于同一个字段内的多个校验器，如果一个短路校验器校验失败后，其他校验器都根本不会继续校验。

(3) 在 com.struts2.actions 包下新建 RegisterAction.properties 文件，内容如下。

```
error.verifyPassword.required=请输入确认密码！
error.verifyPassword.identical=两次输入的密码必须一致！
```

(4) 继续在 com.struts2.actions 包下创建 RegisterAction 类，其内部有一个确认密码属性 verifyPassword 和 Register 实体类对象属性 reg，并对这两个属性进行了注解验证。具体代码如下。

```
package com.struts2.actions;
import com.opensymphony.xwork2.ActionSupport;
import
com.opensymphony.xwork2.validator.annotations.FieldExpressionValidator;
import
com.opensymphony.xwork2.validator.annotations.RequiredStringValidator;
import com.opensymphony.xwork2.validator.annotations.Validations;
import com.opensymphony.xwork2.validator.annotations.ValidatorType;
```

```
import com.opensymphony.xwork2.validator.annotations.VisitorFieldValidator;
import com.struts2.model.Register;
public class RegisterAction extends ActionSupport {
    private Register reg;//注册者实体类对象
    private String verifyPassword;//确认密码
    @Validations(
            requiredStrings=
            {
                    /**
                     * 为 RegisterAction 类中的 verifyPassword 属性配置了必填字符串
                       校验器
                     */
                    @RequiredStringValidator(
                            type=ValidatorType.SIMPLE,
                            fieldName="verifyPassword",
                            key="error.verifyPassword.required"
                    )
            },
            fieldExpressions=
            {
                    /**
                     * 为RegisterAction类中的verifyPassword属性配置了表达式校验器
                     */
                    @FieldExpressionValidator(
                            fieldName="verifyPassword",

    expression="reg.password.equals(verifyPassword)",
                            key="error.verifyPassword.identical"
                    )
            }
    )
    @Override
    public String execute() throws Exception {
        return SUCCESS;
    }
    public Register getReg() {
        return reg;
    }
    //为 Register 类对象 reg 配置 visitor 校验器
    @VisitorFieldValidator(message="提示:")
    public void setReg(Register reg) {
        this.reg = reg;
    }
    /*下面是上面 verifyPassword 属性的 set、get 方法，这里省略*/
}
```

(5)　在 struts-config 文件夹下创建 register.xml 文件，配置 RegisterAction 类。配置如下。

```
<package name ="register" extends ="struts-default" >
        <!-- 配置一个名为 customer 的 Action-->
        <action name ="register" class ="com.struts2.actions.RegisterAction" >
```

```
            <result name="input">/register.jsp</result>
            <result name="success">/success.jsp</result>
        </action>
</package>
```

(6) 把配置好的 register.xml 文件引用至 struts.xml 文件中。

(7) 编辑注册页面 register.jsp 页面，提交转至 RegisterAction 类中 execute()方法所指向的结果页面。register.jsp 页面表单内容如下。

```
<s:form action="register/register.action" method="post" >
                <s:textfield name="reg.username" label="用户名"/>
                <s:password name="reg.password" label="密码"/>
                <s:password name="verifyPassword" label="确认密码"/>
                <s:textfield name="reg.email" label="邮箱"/>
                <s:radio list="%{#{'男':'男','女':'女'}}" name="reg.sex"
label="性别"/>
                <s:textfield name="reg.phone" label="电话"/>
                <s:submit value="注册"/>
</s:form>
```

6.6.4 运行结果

运行 register.jsp 页面，当用户输入有误时，提示错误信息，如图 6-15 所示。

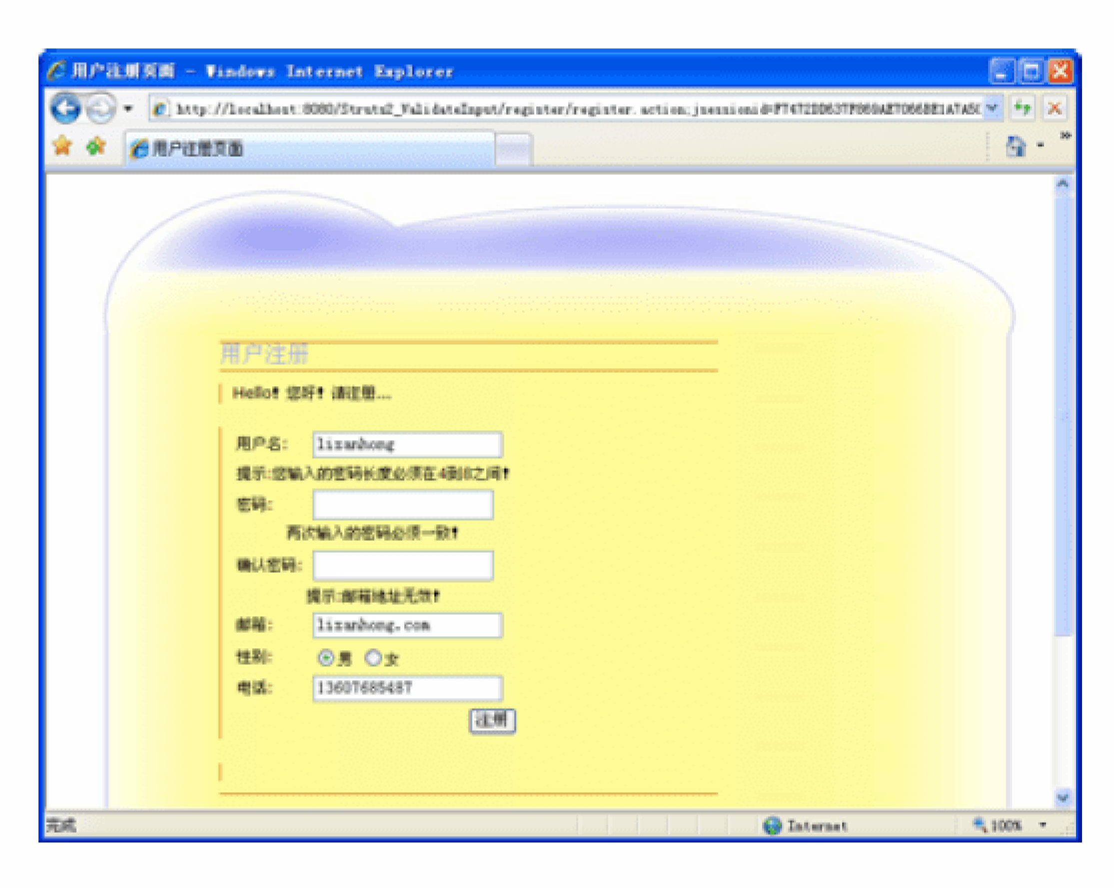


图 6-15 注册提示错误信息

6.6.5 实例分析



源码解析：

在上案例中的 RegisterAction 类中，在 execute()方法上使用了 Validations 注解，集中配置验证规则，并通过各个验证注解的 fieldName 参数指出要验证的字段，这使得在执行 RegisterAction 类中的 execute()方法时会对输入的数据进行校验。

6.7　常见问题解答

6.7.1　Struts 2.1.8 版本是否支持客户端校验



Struts 2.1.8 版本支持客户端校验吗？
网络课堂：http://bbs.itzcn.com/thread-11009-1-1.html

问：Struts 2.1.8 版本支持客户端校验吗？如果该版本支持客户端校验，到底与 Struts 2.1.6 版本有什么配置上的差别呢？

答：这是很多人想知道的问题！我刚刚验证了一下：Struts 2.1.8 完全支持客户端校验，没有任何问题。

如果你配置的 Action 所在的 package 没有指定 namespace 属性，那么 JSP 页面中的 <s:form .../>标签无需任何改变；如果你配置的 Action 所在的 package 指定了 namespace 属性，那么 JSP 页面中的<s:form .../>标签页需要指定 namespace 属性。

还有一点需要注意：不能直接通过 xxx.jsp 页面来使用 Struts 2 的客户端校验，而应该通过一个 Action 来跳转到需要使用客户端校验的页面。

6.7.2　校验器的配置风格都有哪些，它们的校验顺序原则，校验器短路的原则



校验器的配置风格都有哪些？它们的校验顺序原则是什么？校验器短路的原则是什么？
网络课堂：http://bbs.itzcn.com/thread-11010-1-1.html

问：在这章一直提到一个词“校验器的配置风格”，那么校验器的配置风格有哪些呢？而它们的校验顺序原则是什么？在 Struts 2 的内置校验器中不是有一个短路校验器吗？它的原则是什么？

答：Struts 2 提供了两种方式来配置校验规则：字段校验器风格和非字段校验器风格。字段校验器配置风格一般是以<field .../>元素为基本子元素；非字段校验器风格是一种以校验器优先的配置方式，在这种配置方式下，校验规则文件的根元素包含了多个<validator .../>元素，每个<validator .../>元素定义了一个校验规则。

很多时候需要用到短路校验器时，只需要在<validator .../>元素或<field-validator .../>元素中增加 short-circuit="true"即可。



在 Struts 2 的现阶段，客户端校验还不支持短路特性。

校验器的执行顺序有如下原则。

- 所有非字段风格的校验器优先于字段风格的校验器。
- 所有非字段风格的校验器中，排在前面的会先执行。
- 所有字段风格的校验器中，排在前面的会先执行。

校验器短路的原则是。

- 所有非字段校验器是最优先执行，如果某个非字段校验器校验失败了，则该字段上所有字段校验器都不会获得校验的机会。
- 非字段校验器的校验失败，不会阻止其他非字段校验的执行。
- 如果一个字段校验器校验失败后，则该字段下的且排在该校验失败的校验器之后的其他字段校验器不会获得校验的机会。
- 字段校验器永远都不会阻止非字段校验器的执行。

6.7.3 Struts 2 如何显示验证出错信息



Struts 2 如何显示验证出错信息？
网络课堂：http://bbs.itzcn.com/thread-11012-1-1.html

在 Struts 2 中有如下 validate()验证方法。

```
public void validate(){
    if(getUserName() == null ||getUserName().equals("")){
        addFieldError("userNameError","没有输入用户名");
    }
    if(getPassword() == null||getPassword().equals("")){
        addFieldError("passwordError","没有输入密码");
    }
}
```

问：请在 JSP 页面中，如何从 ValueStack 里取出 userNameError？不想用 Struts 2 的<s:fieldError .../>标签，因为它的样式太难控制了，如何让它只显示出错误信息 userNameError 或 passwordError 对应的值呢？

答：从 ValueStack(值栈)中取值有两种方式。

- 直接读取属性名字，例<s:property value= “属性名” />。
- 使用 OGNL 表达式，例<s:property value= “%{属性名}” />。

关于错误提示可以从两个属性中取得。

- errors 属性中取得

```
<s:property value="%{errors.userNameError}"/>
<s:property value="errors.userNameError"/>
```

- fieldErrors 属性中取得

```
<s:property value="%{fieldErrors.userNameError}"/>
<s:property value="fieldErrors.userNameError"/>
```

6.8　习　　题

一、填空题

(1) 手动完成输入校验有三种方式可以实现，分别是在 Action 的 execute()方法中进行校验、在 validateXxx()方法中进行校验和__________。

(2) Struts 2 针对常用的验证器需求，提供了__________个验证器。

(3) 如果想要开发自己的校验器，那么自定义的校验器类必须实现__________接口。

(4) 下面代码表明：在验证 user 属性时，它应该使用__________文件。

```
<field name="user">
        <!-- 使用 visitor 验证程序 -->
        <field-validator type="visitor">
            <!--配置 context 参数-->
            <param name="context">cardid</param>
            <message>Address:</message>
        </field-validator>
</field>
```

二、选择题

(1) 在编辑验证规则文件时，获取国际化消息有两种方式，分别是__________。

A. 通过 key 指定国际化提示信息

B. 通过 ActionSupport 的 getText()方法获取国际化提示信息

C. 通过 value 指定国际化提示信息

D. 通过 Validator 的实现类获取国际化提示信息

(2) 字符串长度验证器(stringlength validator)可以接受 4 个参数，分别为__________。

A. fieldName、min、max、trim

B. fieldName、min、max、shortCircuit

C. fieldName、minLength、maxLength、shortCircut

D. fieldName、minLength、maxLength、trim

(3) 下面这段代码属于__________校验器风格。

```
<field name="name">
    <!-- 指定 name 属性必须满足必填规则 -->
    <field-validator type="requiredstring">
        <param name="trim">true</param>
        <message>必须输入名字</message>
    </field-validator>
</field>
```

A. 混乱校验器风格　　B. 无校验器风格

C. 非字段校验器风格　　D. 字段校验器风格

(4) Struts 2 中与验证相关的注解有__________种。

A. 13　　B. 14

C. 17　　D. 18

三、上机练习

上机练习：完善会员注册功能。

要求：

(1) 在会员注册系统中，对会员姓名配置表达式校验器。正则表达式验证输入的姓名只能是汉字或者字母，不能两者都有，也不能包含任何符号和数字。当用户输入不正确时提示错误信息，如图 6-16 所示。



图 6-16　输入用户信息

(2) 对会员输入的姓名字段加上短路校验器。当用户没有在姓名文本框中输入任何内容时，只提示一条信息，如图 6-17 所示。



图 6-17　短路校验输入数据



第 7 章　Struts 2 中完整的OGNL

内容摘要：

OGNL 是一种用于访问和设置对象数据的强大的表达式语言，它可以自动导航对象图的结构，实现调用对象方法，操作集合，访问类的静态成员，等等。

本章详细介绍了 OGNL 表达式的用法。为了向开发人员提供更好的开发体验，对 Struts 2 在 OGNL 基础上的增强进行了一一讲解。最后针对 OGNL 使用中的常见问题进行解答。

学习目标：

- 理解 OGNL 的三要素。
- 掌握 OGNL 表达式的使用。
- 掌握 OGNL 对集合的操作。
- 掌握 lambda 表达式的使用。
- 理解值栈的概念。
- 掌握 Struts 2 对 OGNL 表达式的增强。

7.1 使用OGNL表达式获取数据

OGNL 是 Object Graph Navigation Language(对象图导航语言)的缩写，是一种表达式语言。使用这种表达式语言，读者可以通过某种表达式语法，存取 Java 对象树中的任意属性、调用 Java 对象树的方法、同时能够自动实现必要的类型转化。如果把表达式看做是一个带有语义的字符串，那么 OGNL 无疑成为了这个语义字符串与 Java 对象之间沟通的桥梁。

视频教学： 光盘/videos/07/ognl_1.avi　　长度：13 分钟
光盘/videos/07/ognl_2.avi　　长度：12 分钟

7.1.1 基础知识——OGNL基础

OGNL 表达式的基础单元就是导航链(Navigation Chain)，简称为链。最简单的链由三个部分构成：属性名、方法调用和数组索引。本节将对 OGNL 的三要素进行基本讲解，希望读者了解 OGNL 的内部结构。在重点讲解 OGNL 表达式使用的同时，读者需要掌握并灵活运用表达式访问操作数据。

1. OGNL三要素

把传入 OGNL 的 API 的三个参数(Expression、Root Object 和 Context)，称之为 OGNL 的三要素。OGNL 的操作实际上就是围绕着这三个参数而进行的。

OGNL 的 API 来自于 OGNL 的静态方法。

```
public static Object getValue( Object exception, Map context, Object root )
throws OgnlException;
public static void setValue( Object tree, Map context, Object root, Object value )
throws OgnlException;
```

1) 表达式(Exception)

表达式是整个 OGNL 的核心，所有的 OGNL 操作都是通过解析表达式后进行的。表达式指定了 OGNL 操作要做的工作。

例如：name、department.name 等都是表达式，表示取 name 或者 department 中的 name 的值。OGNL 支持很多类型的表达式，后面读者将会看到更多。

2) 根对象(Root Object)

根对象可以理解为 OGNL 要操作的对象，在表达式规定了“要做的工作”以后，需要指定工作的操作对象。

比如指定 user 就是根对象，表达式采用 name。这就意味着，需要对 user 这个对象去取 name 这个属性的值。代码如下。

```
Ognl.getValue(Ognl.parseExpression("name"), user);
```

3)　上下文环境(Context)

有了表达式和根对象，实际上已经可以使用 OGNL 的基本功能。例如，根据表达式对根对象进行取值或者设值工作。

不过在 OGNL 的内部，所有的操作都会在一个特定的环境中运行，这个环境就是 OGNL 的上下文环境(Context)。即这个上下文环境(Context)，规定 OGNL 的操作地点。

OGNL 的上下文环境是一个 Map 结构，称为 OgnlContext。上面提到的根对象(Root Object)，事实上也会被加入到上下文环境中去，并且这将作为一个特殊的变量进行处理。

OgnlContext 不仅提供了 OGNL 的运行环境。在这其中，还能设置一些自定义的 parameter 到 Context 中，以便在进行 OGNL 操作的时候能够方便地使用这些 parameter。



在访问 parameter 时，需要使用#作为前缀才能进行。还有刚提到的 Root Object 作为一个特殊的变量进行处理，具体就表现为，针对根对象的存取操作的表达式是不需要增加#符号进行区分的。

2. OGNL表达式

OGNL 支持各种复杂的表达式。但是最基本的表达式是将对象的引用值用“. ”串联起来。从左到右，每一次表达式计算返回的结果成为当前对象，后面部分接着在当前对象上进行计算，一直到全部表达式计算完成，返回最后得到的对象。OGNL 则针对这条基本原则进行不断的扩充，从而使之支持对象树、数组、容器的访问等操作。

下面是一些常用的 OGNL 表达式。

1)　常量

OGNL 支持的常量除了包含 Java 语言的常量类型外，还提供了自有的常量类型，方便 OGNL 的使用。

OGNL 支持的所有常量类型如下所示。

(1)　字符串常量。使用单引号或双引号括起来的字符串，例如：'Welcome to China'，"Welcome to China"。在 Java 中不可以使用单引号来界定字符串常量，而 OGNL 中可以。不过如果是单个字符的字符串常量，则必须使用双引号来界定，如："H"。OGNL 的字符串也支持转义序列，例如：要在 JSP 页面中输出“You said，"Welcome to China"。”，代码如下。

```
<s:property value="'You said，\"Welcome to China\"。'"/>
```

(2)　字符常量。用单引号括起来的字符。例如：'C'。注意，不能使用双引号，否则将被认为是字符串常量。

(3)　数值常量。除了 Java 中的 int、long、float 和 double 外，OGNL 中还可以使用“b”或“B”后缀指定 BigDecimal 常量，用“h”或“H”后缀指定 BigInteger 常量。例如：345(int 常量)、345l(long 常量)、345.34f(float 常量)、123.45(double 常量)、23b(BigDecimal 常量)、123h(BigInteger 常量)。

(4)　布尔常量。布尔常量只有两个值：true 和 false。

(5)　null 常量。表示空值。

2) 基本对象树的访问

对象树的访问就是通过使用“.”号将对象的引用串联起来进行。如下所示。

```
name、department.name、user.department.factory.manager.name
```

3) 对容器变量的访问

对容器变量的访问，通过#符号加上表达式进行。如下所示。

```
#name、#department.name、#user.department.factory.manager.name
```

4) 操作符号

OGNL 表达式中能使用的操作符基本跟 Java 里的操作符一样，除了能使用+、-、*、/、++、--、==、!=、=等操作符之外，还能使用 mod、in、not in 等。与 Java 相同的操作符(+、-、*、/等)就不做详细讲解了，下面对 OGNL 特有的操作符进行讲解。

(1) 逗号(，)或序列操作符。OGNL 的逗号操作符是借鉴 C 语言中的。逗号被用于分隔两个或多个独立的表达式，整个表达式的值是最后一个子表达式的值。例如：name，#manager.name。

第一个表达式 name 和第二个表达式#manager.name 依次被计算，整个表达式的值是第二个表达式的值。

(2) 花括号({})操作符。花括号({})操作符用于创建列表。使用花括号将元素括起来，元素之间使用逗号分隔，例如表达式{"Rose","Bob","Marry"}，创建了带有三个元素的列表。

(3) in 和 not in 操作符。in 和 not in 操作符用于判断一个值是否在集合中，例如：name in{null,"sky"}||name。

5) 访问 JavaBean 的属性

访问 JavaBean 属性的表达式是非常常用的。假如有一个 employee 对象作为 OGNL 上下文的根对象，那么就对应于下列的表达式。

(1) name。相当于 Java 代码：employee.getName()。

(2) address.country。相当于 Java 代码：employee.getAddress.getCountry()。

6) 容器、数组、对象

OGNL 支持对数组和 ArrayList 等容器的顺序访问。如下所示。

```
group.users[0];
```

同时，OGNL 支持对 Map 的按键值查找。如下所示。

```
#session[mySessionPropKey]
```

不仅如此，OGNL 还支持容器构造的表达式。如下所示。

```
{"green","red","blue"}        //构造一个 List
#{"key1":"value1","key2":"value2","key3":"value3"}       //构造一个 Map
```

可以通过任意类对象的构造函数进行对象新建。如下所示。

```
new java.net.URL("http://localhost")
```

7)　对静态方法或变量的访问

引用类的静态方法和字段，它们的表达方式是一样的@class@member 或者@class@method (args)。如下所示。

```
@com.java.core.Resource@ENABLE      //调用静态字段
@com.java.core.Resource@getAllResources()  //调用静态方法
```

8)　方法调用

可以直接通过类似 Java 的方法调用方式进行，也可以通过传递参数的方式。如下所示。

```
user.getName();
group.users.size();
group.containsUser(requestUser);
```

9)　调用构造方法

OGNL 也支持对构造方法的调用，用于创建一个新的对象。可以像在 Java 中一样使用 new 操作符来创建一个对象，不同的是，必须使用完整的限定类名(带包名的类名)，如下所示。

```
<s:property value="new com.struts2.action.model.Dog()"/>  //创建一个 Dog 对象
```

7.1.2　实例描述

星期天上网闲逛时，有人在论坛中问：什么是 OGNL，还要顺便给个例子。我正好没事，就帮他写了个例子，主要就是使用 OGNL 表达式获取一些简单数据。这个例子比较适合初学者拿来练习。

7.1.3　实例应用

【例 7-1】 使用 OGNL 表达式获取数据。

(1)　首先新建一个项目 struts2_7，然后在项目的 src 下新建一个 struts2.action 包，最后在该包下新建一个 Person.java 类，该实体类中声明了个人编号 no、年龄 age、名字 name 和性别 sex。代码如下所示。

```
public class Person {
    private String no;      //个人编号
    private int age;        //年龄
    private String name;    //名字
    private String sex; //性别
    public Person(){}
    public Person(String no, String sex) {
        super();
        this.no = no;
        this.sex = sex;
```

```
    }
    public String getNo() {
        return no;
    }
    public void setNo(String no) {
        this.no = no;
    }
    //下面是 age、name 和 sex 属性的 get、set 方法，在此省略。
    ..........................................................
}
```

(2) 定义一个 OgnlAction，通过 ActionContext 上下文获取 request、session 和 application，并分别为它们添加参数信息，并返回 ognl 结果，代码如下所示。

```
public class OgnlAction {
    private String name;          //声明 name 属性
    private List list = new ArrayList();        //声明一个 List 列表

    public List getList() {
        return list;
    }
    public void setList(List list) {
        this.list = list;
    }
    public String execute(){
        name = "张三";                      //为 name 属性设置值
        Person p = new Person("1001","男"); //为 list 属性添加一个 Person 对象
        list.add(p);                        //将 Person 对象 p 添加到 list 中
        Map request = (Map)ActionContext.getContext().get("request");
        //获取 request，并在 request 中添加 name
        HttpServletRequest request1 = ServletActionContext.getRequest();
        request1.setAttribute("name", "request value");
        //  获取 session，并添加信息
        Map session = ActionContext.getContext().getSession();
        session.put("name", "session value");
        //获取 application，并添加信息
        Map application = ActionContext.getContext().getApplication();
        application.put("name", "application value");
        return "ognl";       //返回 ognl
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
```

```
        this.name = name;
    }

}
```

(3) 在项目的 src 目录下新建一个 struts.xml 文件，打开该文件在该文件中添加 OgnlAction 的配置，代码如下所示。

```
<struts>
    <constant name="struts.action.extension" value="do"/>
    <!-- 注意 namespace 作用 -->
    <package name="myfirst1" extends="struts-default">
        <action name="ognl" class="struts2.action.OgnlAction">
            <result name="ognl">/ognl.jsp</result>     <!-- 返回 ognl 跳转到
ognl.jsp 页面 -->
        </action>
    </package>
</struts>
```

(4) 创建 ognl.jsp 页面，用来通过 ognl 表达式获取前面在 OgnlAction 中放在 request、session 和 application 中的数据。代码如下所示。

```
<body>
    <h1>${name}</h1>
    <hr>
    获取 Action 属性：<s:property value="name"/>
    <br>
    获取 Reqeust 属性：<s:property value="#request.name"/><br>
    获取 Session 属性：<s:property value="#session.name"/><br>
    获取 Application 属性：<s:property value="#application.name"/><br>
    <hr>
    获取 Action 中 List 属性信息
    <s:property value="list[0].no"/>
    <s:property value="list[0].sex"/>
    <hr>
    方法调用<s:property value="name.length()"/><br>
    静态属性：<s:property value="@java.lang.Math@PI"></s:property>
</body>
```

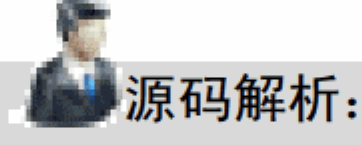
7.1.4　运行结果

打开 IE 浏览器，在地址栏中输入“http://localhost:8080/Struts2_7/ognl.do”，OgnlAction 请求成功之后，将跳转到 ognl.jsp 页面，在该页面中显示 OGNL 表达式获取的数据信息，执行效果如图 7-1 所示。



图 7-1　OGNL表达式获取的数据信息

7.1.5　实例分析

**源码解析：**

本实例中我们定义了一个 OgnlAction，通过 ActionContext 上下文中获取 request、session 和 application 对象，并为它添加了参数 name。为 OgnlAction 的 name 属性赋值为“张三”，并为 List 列表添加了一个 Person 对象。当请求 OgnlAction 成功时，跳转到 ognl.jsp 页面上，在该页面上通过 OGNL 表达式分别获取 request、session 和 application 对象中的参数值，并获取 OgnlAction 的 name 属性值“张三”，List 列表中的 Person 对象信息。

7.2　人员集合的操作

在 OGNL 表达式应用中，对集合的操作使用是非常频繁的，本节将详细讲解 OGNL 对集合的操作。

视频教学：光盘/videos/07/collection.avi　　　　长度：10 分钟

7.2.1　基础知识——OGNL对集合的操作

OGNL 提供了对 Java 集合 API 非常好的支持，创建集合并对其进行操作是 OGNL 的一个基本特性。本节将看一下创建并访问集合的方式，以及如何根据集合的内容进行投影和选择。

1. 创建集合

1)　创建列表

使用花括号将元素包含起来，元素之间使用逗号分隔。如下所示。

```
{'Marry','Jack','Rose'}
```

接下来定义一个元素类型是 String，包含两个元素的 List 对象，代码如下所示。

```
List list = new ArrayList(2);
list.add("海燕");
list.add("豚鼠");
```

通过索引来访问列表，如：list[0]、{'Marry','Jack','Rose'}[1]。

2)　创建数组

OGNL 中创建数组与 Java 语言中创建数组类似。例如：表达式：new String[]{"hello","你好","nihao"}创建一个 String 类型的数组，由三个字符串值“hello”、“你好”、“nihao”组成。

可以通过索引来访问，如：array[0]。Java 中数组有一个 length 属性可以获取数组的长度。在 OGNL 中也可以直接访问，例如：array.length。

3)　创建 Map

Map 使用特殊的语法来创建，代码如下。

```
#{"key":"value",……}
```

如果想指定创建的 Map 类型，可以在左花括号前制定 Map 实现类的类名。代码如下所示。

```
#@java.util.LinkedHashMap@{"key":"value",......}
```

Map 通过 key 来访问，如 map["key"]或 map.key。

2．集合的伪属性

Java 集合 API 提供了很多常用的方法，如：size()、isEmpty()等，但这些方法的命名并不符合 JavaBean 对于属性访问器方法的命名要求(即 getXxx()和 setXxx())，不能像访问 JavaBean 属性一样来调用这些方法。为了方便对 Java 集合 API 的方法调用，OGNL 提供了一些伪属性，使得可以按照属性的访问方式来调用集合中的方法。

OGNL 提供的集合伪属性如表 7-1 所示。

表 7-1　特殊的集合伪属性

集合类型	伪 属 性	OGNL 表达式	Java 代码
List、Set、Map	size，isEmpty	list.size，set.isEmpty	list.size()，set.isEmpty()
List	iterator	list，iterator	list.iterator()
Map	keys，values	map.keys，map.values	map.keySet()，map.values()
Set	iterator	set.iterator	set.iterator()
Iterator	next，hasNext	iter.next，iter.hasNext	iter.next()，iter.hasNext()
Enumeration	next，hasNext，nextElement，hasMoreElements	enum.next，enum.hasNext	enum.nextElement()，enum.hasMoreElements()



Enumeration 的两对伪属性是同义的，即 next 等价于 nextElement，hasNext 等价于 hasMoreElements。

3．投影与选择

OGNL 支持类似数据库中的投影(projection)和选择(selection)。

1) 投影

投影就是选出集合中每个元素的相同属性组成新的集合，类似于关系数据库的字段操作。投影操作语法所下。

```
collection.{XXX}
```

其中 XXX 是这个集合中每个元素的公共属性。代码如下。

```
group.userList.{username}        //将获得某个 group 中的所有 user 的 name 的列表
```

2) 选择

选择就是过滤满足 selection 条件的集合元素，类似于关系数据库的纪录操作。选择操作的语法为如下。

```
collection.{X YYY}
```

其中 X 是一个选择操作符，后面则是选择用的逻辑表达式。选择操作符有三种。

- ?：选择满足条件的所有元素。
- ^：选择满足条件的第一个元素。
- $：选择满足条件的最后一个元素。

例子如下所示。

```
#employees.{?#this.salary>3000}
```

将返回薪水大于 3000 的所有员工的列表。

```
#employees.{^#this.salary>3000}
```

将返回第一个薪水大于 3000 的员工的列表。

```
#employees.{$#this.salary>3000}
```

将返回最后一个薪水大于 3000 的员工的列表。

7.2.2　实例描述

作为程序开发者，整天都有忙不完的项目。还好现在手头的项目马上就完工了。看着自己的累累硕果，心里还是很开心的，辛苦一点也值得。在这个项目中，使用 Struts 2 的 OGNL 比较多，主要就是对 OGNL 集合的操作，做这个项目使我对 OGNL 集合的操作有了深入的理解。今天抽空写了个例子，把我的收获与读者分享一下。

7.2.3　实例应用

【例 7-2】 人员集合的操作。

(1) 本实例引用了 7.1.4 节实例中的 Person 类，并在该类中添加了一个带参构造方法，代码如下所示。

```
public Person(String no,int age,String name,String sex){
        this.no = no;         //编号
        this.age = age;       //年龄
        this.name = name;     //姓名
        this.sex = sex;       //性别
}
```

(2) 在项目 src/struts2/action 目录下，新建一个 OgnlPersonAction，该 Action 中定义了一个 List 用于存储 Person 对象的集合，在 request、session 和 application 中放入了 username 参数。代码如下所示。

```
public class OgnlPersonAction extends ActionSupport{

    private List<Person> persons;
    public void setPersons(List<Person> persons) {
        this.persons = persons;
    }
    public List<Person> getPersons() {
        return persons;
    }
    public String execute() {
        HttpServletRequest request = ServletActionContext.getRequest();
        request.setAttribute("userName","Max From request");
        //获取 session，并添加信息
        Map session = ActionContext.getContext().getSession();
        session.put("userName", "Max From session");
        //获取 application，并添加信息
        Map application = ActionContext.getContext().getApplication();
        application.put("userName", "Max From application");
        persons = new LinkedList<Person>();
        //向 persons 集合中添加 Person 对象
        persons.add(new Person("2010-619678", 25,"马海涛","男"));
        persons.add(new Person("2010-007867", 42,"赵海燕","女"));
        persons.add(new Person("2010-633610", 68,"蔡文静","女"));
        persons.add(new Person("2010-527341", 22, "郭小刚","男"));
        persons.add(new Person("2010-605350", 37,"赵文卓","男"));
        return "ognl";
    }
}
```

(3) 打开 src 目录下的 struts.xml 文件，添加 OgnlPersonAction 的配置，代码如下所示。

```
<package name="Struts2_OGNL_DEMO" extends="struts-default">
    <action name="ognlperson" class="struts2.action.OgnlPersonAction">
        <result name="ognl">/ognlperson.jsp</result>
    </action>
</package>
```

(4) 新建一个 onglperson.jsp 页面，在该页面中首先访问 OGNL 上下文和 Action 上下文取出 request、session、application 和 attr 中的参数 username，然后通过过滤和投影(projecting)集合，在页面中显示出年龄大于 30 的人员，最后构造 Map 集合，使用 OGNL 取出 Map 集合中数据，代码如下所示。

```
<body>
    <h3>访问 OGNL 上下文和 Action 上下文</h3>
    <p>request.userName: <s:property value="#request.userName" /></p>
    <p>session.userName: <s:property value="#session.userName" /></p>
    <p>application.userName: <s:property value="#application.userName" /></p>
    <p>attr.userName: <s:property value="#attr.userName" /></p>
    <hr />
    <h3>用于过滤和投影(projecting)集合</h3>
    <p>年龄大于 30 岁的人</p>
    <ul>
        <s:iterator value="persons.{?#this.age > 30}">
            <li><s:property value="name" /> 年龄: <s:property value="age"
/></li>
        </s:iterator>
    </ul>
    <p>赵海燕的年龄: <s:property value="persons.{?#this.name=='赵海燕
'}.{age}[0]"/></p>
    <h3>构造 Map</h3>
    <s:set name="foobar" value="#{'语文':'86','数学':'100'}"/>
    <p>语文成绩: <s:property value="#foobar['语文']" /></p>
</body>
```

7.2.4 运行结果

完成实例代码后，打开 IE 浏览器，在地址栏中输入“http://localhost:8080/Struts2_7/ognlperson.action”，执行效果如图 7-2 所示。

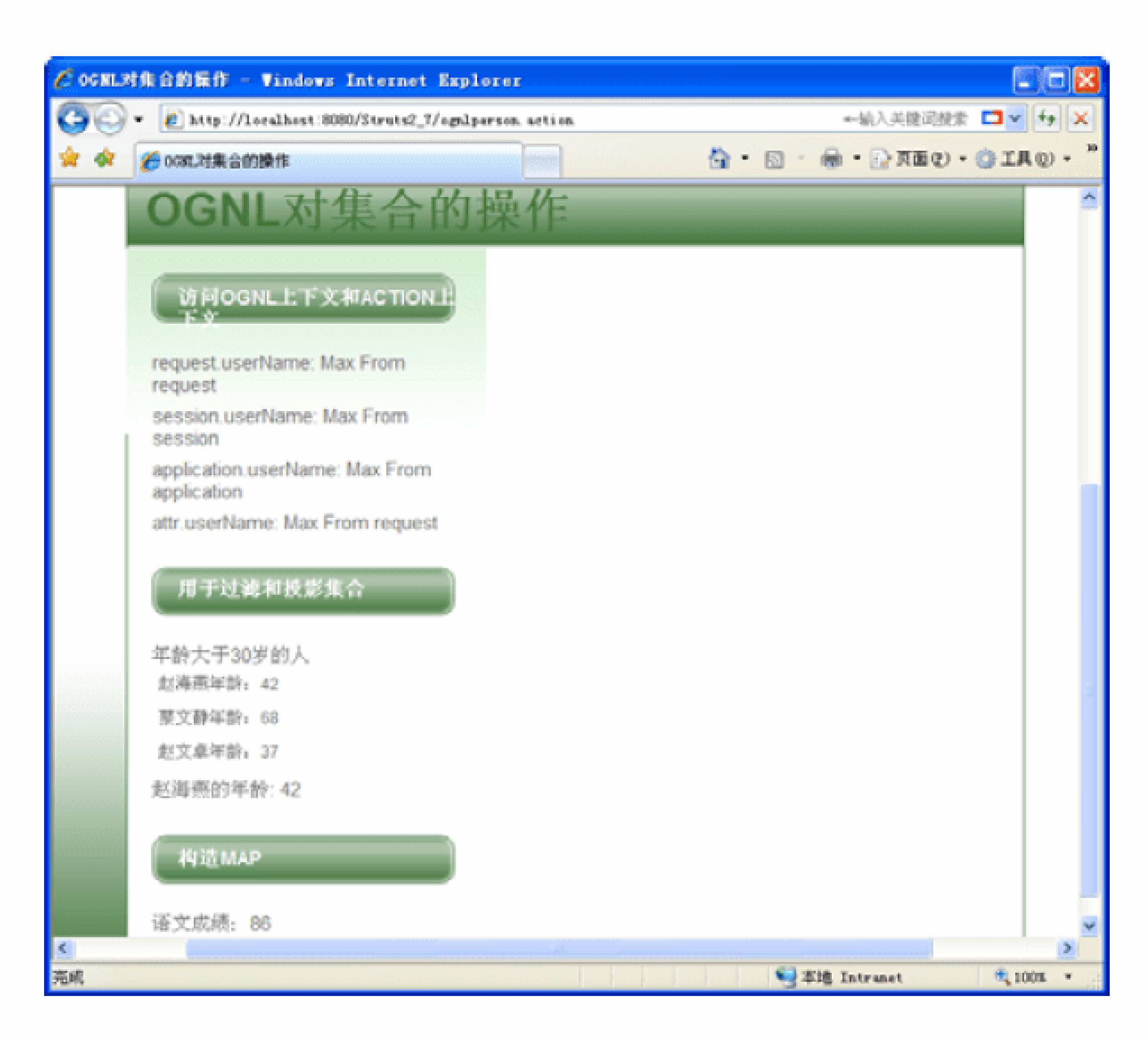


图 7-2　人员集合的操作

7.2.5　实例分析


源码解析：

上述实例主要通过一个 OgnlPersonAction，在 Action 中声明了一个 List 集合列表用来存储 Person 对象，通过使用表达式“persons.{?#this.age > 30}”依次迭代获取集合中年龄大于 30 岁的人，将这些人的信息显示在页面中。

7.3　公司员工性别调查

“lambda 表达式”是一个匿名函数，它可以包含表达式和语句，并且可用于创建委托或表达式目录树类型。OGNL 中也提供了一种 lambda 表达式语法。本节将讲解它的具体使用情况。


视频教学： 光盘/videos/07/lambda.avi　　　　**长度：** 10 分钟

7.3.1　基础知识——lambda表达式

OGNL 有一个简化的 lambda 表达式语法，能够让读者写一些简单的功能。定义 lambda 表达式的语法如下。

```
:[表达式...]
```

OGNL 中的 lambda 表达式只能使用一个参数，但这个参数可以通过#this 变量来引用。OGNL 将 lambda 表达式看作是常量。举个例子，声明一个使用递归来计算阶乘的函数，然后调用它，代码如下。

```
#fact = :[#this<=1?1:#this*#fact(#this-1)],#fact(30H)
```

lambda 表达式是方括号([])中的部分。#this 变量代表表达式的参数，它的初始值是 30H(后缀 H 表示这是一个 BigInteger 常量)，每一次递归调用表达式都将参数的值减 1。整个 OGNL 表达式是一个使用了逗号(，)操作符的表达式，它的值就是调用 lambda 表达式的值。

声明一个计算斐波那契数列的函数，代码如下。

```
#fib=:[#this==0?0:#this==1?1:#fib(#this-2)+#fib(#this-1)]  //#this 变量代表表达式的参数
```

上面仅定义了一个 lambda 表达式。如何调用这个表达式呢？代码如下所示。

```
#fib(11)    //调用了上述表达式，#this 变量的初始值为 11
```

7.3.2 实例描述

在实际开发中我经常使用 OGNL，不过 OGNL 中的 lambda 表达式，我用得较少。前天我没事在那儿研究了一下，突然发现 lambda 表达式原来这么强大。原来要费很大工夫才能实现的功能，现在只要短短一个小式子就可以实现了。本实例将使用 lambda 表达式实现一个将 int 类型的性别字段值，通过简单判断，在页面上显示成“男”和“女”字符串。

7.3.3 实例应用

【例 7-3】 公司员工性别调查。

(1) 首先抽象封装一个员工类 Employee，声明员工编号(no)、员工名称(name)、年龄(age)、性别(sex)和职位(job)等属性，将属性 sex 定义为 int 类型(属性值为 1：女、2：男)，分别给出各属性的 get 和 set 方法，声明一个 Employee 的带参构造函数，实现代码如下所示。

```
public class Employee {
    private String no;  //编号
    private String name;    //名称
    private int age;        //年龄
    private int sex;        //性别
    private String job;     //职位
    public Employee(String no ,String name,int age,int sex,String job){
    //构造函数
        this.no = no;
        this.name = name;
        this.age = age;
        this.sex = sex;
        this.job = job;
    }
```

```
    public String getNo() {
        return no;
    }
    public void setNo(String no) {
        this.no = no;
    }
    //下面是员工名称、年龄、性别和职位属性的 get 和 set 方法，在此省略。
}
```

(2) 创建一个 EmployeeAction，声明一个 List 用来存储 Employee 对象，重新 execute()方法，在该方法中初始化员工列表，代码如下所示。

```
public class EmployeeAction extends ActionSupport{
    private List<Employee> employees;          //公司员工信息列表
    public List<Employee> getEmployees() {     //获取公司员工信息列表
        return employees;
    }
    public void setEmployees(List<Employee> employees) {   //设置员工信息列表
        this.employees = employees;
    }
    public String execute() {
        employees = new LinkedList<Employee>();        //初始化员工信息列表
        //向列表中添加员工对象。
        employees.add(new Employee("20051025","李梅",28,1,"财政主管"));
        employees.add(new Employee("20010509","张瑞",34,2,"市场部经理"));
        employees.add(new Employee("20091015","黄炳",38,2,"行政部经理"));
        employees.add(new Employee("20100229","周静",23,1,"设计部职员"));
        employees.add(new Employee("20030416","郭晓斌",25,2,"技术部组长"));
        return "ognl";
    }
}
```

(3) 在项目 src 目录下 struts.xml 文件中添加 EmployeeAction 的配置，代码如下所示。

```
<package name="Struts2_Lambda_DEMO" extends="struts-default">
    <action name="employee" class="struts2.action.EmployeeAction">
        <result name="ognl">/employee.jsp</result>
    </action>
</package>
```

(4) 新建一个 employee.jsp，该页面用来显示公司员工的性别普查结果，前面封装的 Employee 类中将性别属性 sex 声明为 int 类型(1：女、2：男)，这里可以使用本节讲的 lambda 表达式将 int 类型的性别值转换成字符串“男”和“女”，显示在页面上，使用户看起来更加清晰明朗，代码如下所示。

```
<s:iterator value="employees">
    <li>
        <s:property value="name" />性别：
```

```
<!--lambda 表达式将当前员工的性别值传入 conv 函数，判断值为 1 显示“女”，值为 2 显示“男”，其他显示空白-->
        <s:property value="#conv =:[#this==1?'女':#this==2?'男':''],
#conv(#this.sex)" />
    </li>
</s:iterator>
```

7.3.4 运行结果

启动服务器之后，打开 IE 浏览器，在地址栏中输入“http://localhost:8080/Struts2_7/employee.action”，执行成功后显示员工性别调查结果，如图 7-3 所示。



图 7-3　员工性别调查

7.3.5 实例分析



源码解析：

本实例封装了一个 Employee 类，声明了编号(no)、名称(name)和性别(sex)等属性。其中属性 sex 为 int 类型(1：女、2：男)，当把员工性别信息显示在页面上时，通过 lambda 表达式#conv =:[#this==1?'女':#this==2?'男':''], #conv(#this.sex)创建了一个 conv 函数，当判断员工性别属性值为 1，在页面中显示“女”，为 2 显示“男”。

7.4 获取建材信息

Struts 2 对 OGNL 表达式的增强：值栈、[N]语法、top 关键字、访问静态成员、值栈中的 Action 实例和 Struts 2 中的命名对象。本节将对这些增强一一进行讲解。



视频教学： 光盘/videos/07/struts2_ognl.avi　　　　长度：11 分钟

7.4.1　基础知识——Struts 2 对OGNL表达式的增强

Struts 2 在 OGNL 之上提供的最大附加特性就是支持值栈(ValueStack)。在 OGNL 上下文中只能有一个根对象，Struts 2 的值栈则允许存在许多虚拟根对象。

1．值栈(ValueStack)

Struts 2 将 OGNL 上下文设置为 Struts 2 中的 ActionContext(内部使用的仍然是 OgnlContext)，并将值栈作为 OGNL 的根对象。值栈与正常的栈很类似，也遵循后进先出的规则，可以在值栈中放入、删除和查询对象，值栈是 Struts 2 的核心。

值栈通过一个接口(com.opensymphony.xwork2.util.ValueStack)来定义，对应的实现类是 com.opensymphony.xwork2.util.OgnlValueStack。

顺着值栈，框架在 ActionContext 中还放置了其他对象，包括表示 application、session 和 request 的 Map 对象。这些对象共存于 ActionContext 中，靠在值栈(OGNL 根对象)的旁边，如图 7-4 所示。

图 7-4　值栈

直接访问 OGNL 上下文中的根对象时，不需要使用“#”来标记。而引用上下文中的其他对象时则需要使用“#”来标记。由于值栈是上下文中的根对象，因此可以直接访问。那么值栈中的对象又该如何访问呢？Struts 2 提供了一个秘密武器——OGNLPropertyAccessor(ognl.ObjectPropertyAccessor 是一个接口，Struts 2 提供的实现类是一个静态的内部类 OgnlValueStack.ObjectAccessor)。它可以自动查询栈内的所有对象(从栈顶到栈底)，直到找到一个所查找的属性的对象。也就是说，对于值栈中的任何对象都可以直接访问，而不需要使用“#”。

举个例子，假设值栈中有两个对象：Student 和 Class，两个对象都有 name 属性，Student 有学号属性 number，　Class 有学员总数属性 stu_num。Class 先入栈，Student 后入栈，位于栈顶，如图 7-5 所示。

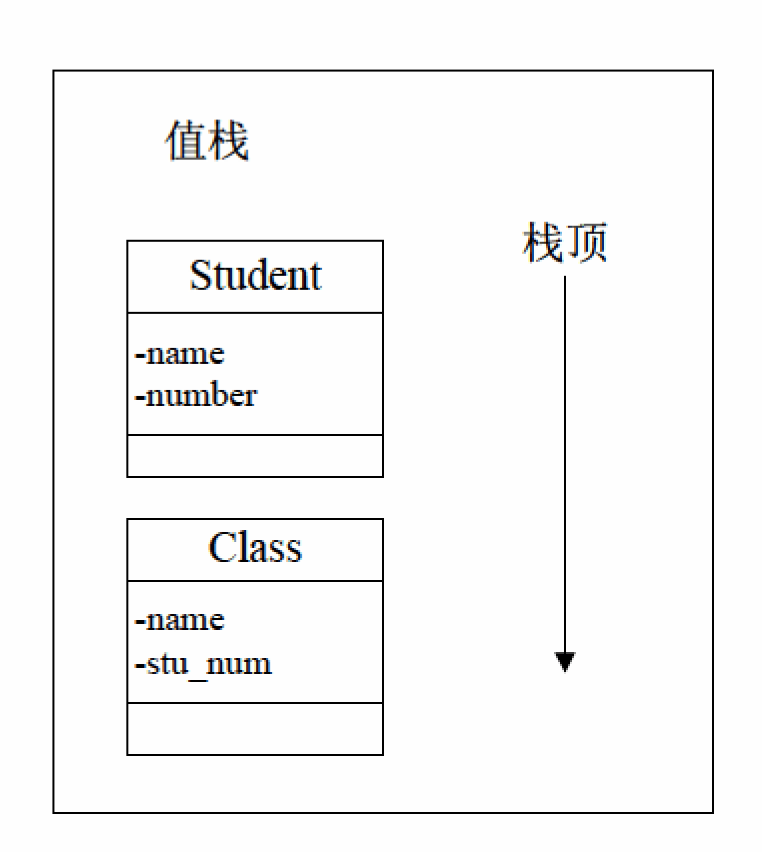


图 7-5　一个包含Student和Class对象的简单值栈

对于表达式 name，访问的就是 Student 对象的 name 属性，因为 Student 对象位于栈顶；表达式 stu_num，访问的就是 Class 对象的 stu_num 属性。访问值栈中的对象属性或方法，无须指明对象，也不用“#”，就好像值栈中的对象都是 OGNL 上下文的根对象一样。这就是 Struts 2 在 OGNL 基础上做出的改进。

2．[N]语法

如果想要访问 Class 的 name 属性，应该如何写表达式呢？可以使用[N].xxx(N 是从 0 开始的整数)这样的语法来指定从哪一个位置开始向下查找对象的属性，表达式[1].name 访问的就是 Class 对象的 name 属性。

在使用[N].xxx 语法时，要注意位置序号的含义，它并不是表示“获取栈中索引为 N 的对象”，而是截取从位置 N 开始的部分栈。假设现在栈中有三个对象：Object0、Object1 和 Object2，Object0 和 Object2 都有 name 属性，如图 7-6 所示。

表达式 name 访问的是 Object0 的 name 属性，[1]是一个包含了 Object1 和 Object2 的部分栈，由于只有 Object2 有 name 属性，所以[1].name 访问的是 Object2 的 name 属性。如图 7-7

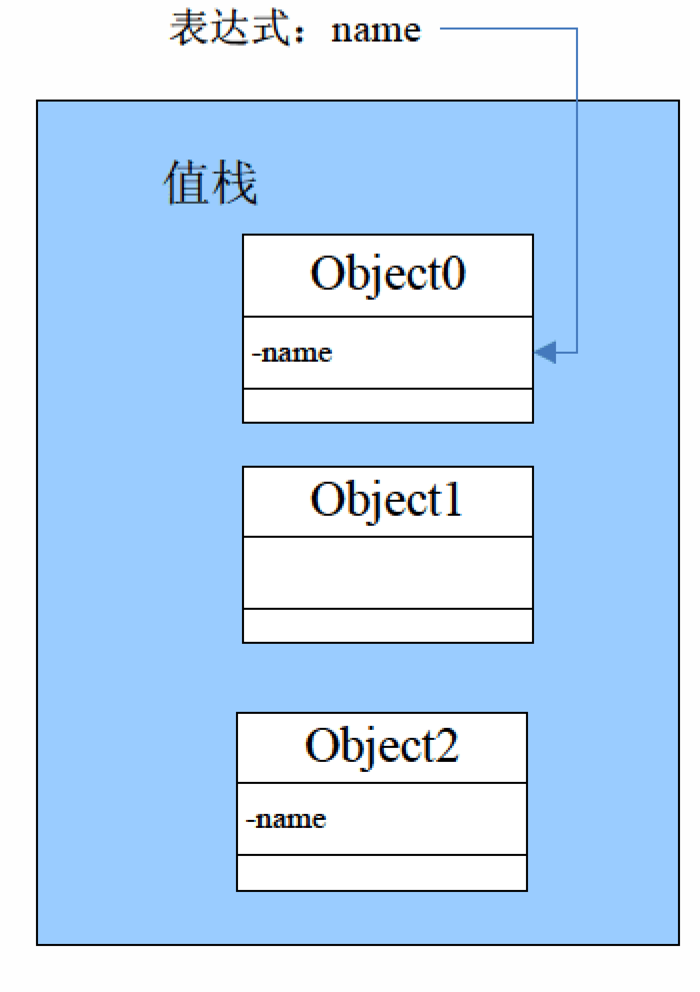


图 7-6　值栈中包含三个对象

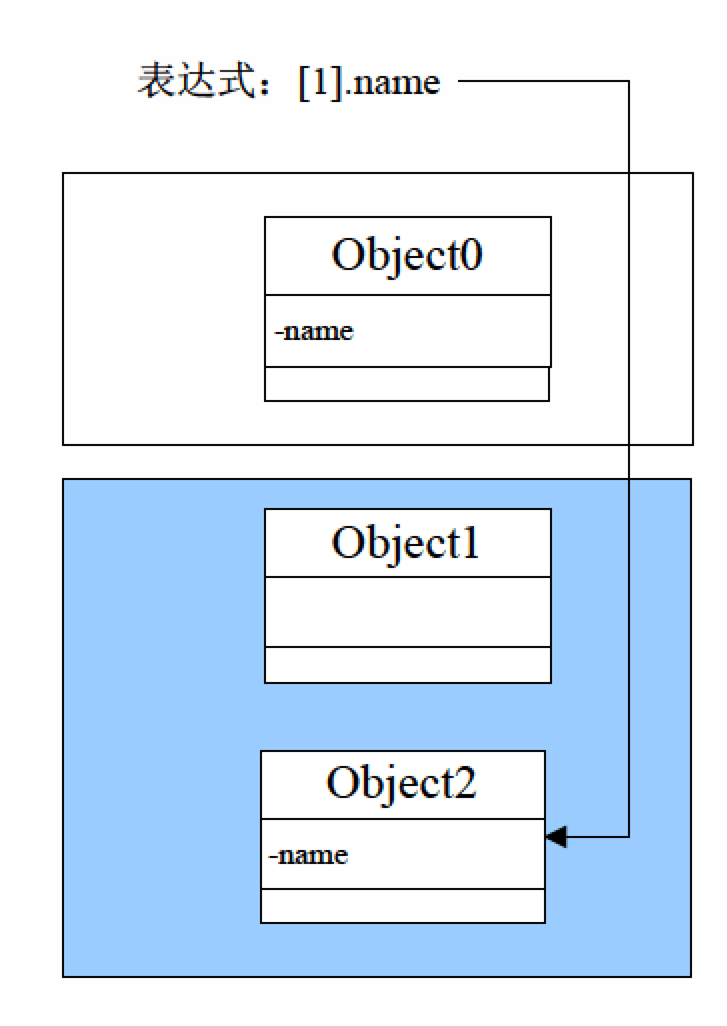


图 7-7　访问[1].name

3．top关键字

通过上面的学习，读者对访问栈中对象的属性和方法应该没有问题了。但能否直接访问栈中的对象呢？Struts 2 引入了一个新的关键字：top，用于获取栈顶的对象。对于如图 7-6 所示的栈，表达式 top 获取的就是 Object0 这个对象本身。结合[N].xxx 语法，可以获取栈中任意位置的对象。例如[0].top 获取 Object0(等同于 top)，[1].top 获取 Object1，[2].top 获取 Object2，[2].top.name 访问 Object2 中的 name 属性。

4．访问静态成员

除了使用标准的 OGNL 表达式访问静态字段和静态方法外，Struts 2 还允许不指定完整的类名，而是通过 vs 前缀来调用保存在栈中的静态字段和静态方法。例如下面所示。

```
@vs@FOO_PROPERTY
@vs@someMethod()
@vs1@someMethod()
```

vs 表示 ValueStack，如果只有 vs，那么将使用栈顶对象的类；如果在 vs 后面跟上一个数字，那么将使用栈中指定位置处的对象类。

Struts 2 中提供了一些访问静态成员的方式，但是默认是关闭的。需要使用时，可以在 Struts 2 的配置文件中添加入如下设置代码。

```
<constant name="struts.ognl.allowStaticMethodAccess" value="true"/>
```

设置之后就可以用以下的表达式来访问静态成员了。代码如下所示。

```
<s:property value="@com.struts2.util.DBUtil@getName()"/>       //访问静态方法
<s:property value="@ com.struts2.util.DBUtil@Index_Title"/>    //访问静态常量
```

5．值栈中的Action实例

Struts 2 框架总是把 Action 实例放置在栈顶。因为 Action 在值栈中，而值栈又是 OGNL 的根，所以引用 Action 的属性可以省略“#”标记，这也是在结果页面中直接访问 Action 的属性的原因，代码如下。

```
<s:property value="name"/>
```

<s:property>标签输出栈顶的 Action 实例的 name 属性值。但是，如果访问 ActionContext 中的其他对象，则必须使用“#”号，以便让 OGNL 知道不是在根对象(即值栈)中查看，而是查看 ActionContext 中的其他对象。

6．Struts 2 中的命名对象

Struts 2 还提供了一些命名对象，这些对象没有保存在值栈中，而是保存在 ActionContext 中，因此访问这些对象需要使用“#”标记。这些命名对象都是 Map 类型。如：parameters、request、session，等等。下面对它们进行详细讲解。

1) parameters

parameters 用于访问请求参数。如：#parameters['id']或#parameters.id，相当于调用了 HttpServletRequest 对象的 getParameter()方法。



parameters 本质上是一个使用 HttpServletRequest 对象中的请求参数构造的 Map 对象，一旦对象被创建(在调用 Action 实例之前就已经创建好了)，它和 HttpServletRequest 对象就没有了任何关系。

2) request

request 用于访问请求属性。如：#request['user']或#request.user，相当于调用了 HttpServletRequest 对象的 getAttribute()方法。

3) session

session 用于访问 session 属性。如：#session['user']或#session.user，相当于调用了 HttpSession 对象的 getAttribute()方法。

4) application

application 用于访问 application 属性。如：#application['user']或#application.user，相当于调用了 ServletContext 的 getAttribute()方法。

5) attr

如果 PageContext 可用，则访问 PageContext，否则依次搜索 request、session 和 application 对象。

7.4.2 实例描述

前段时间，一个朋友问我 Struts 2 对 OGNL 都有哪些增强，如何使用它们。当时被问的丈二和尚摸不着头脑，因为关于这个我还真没总结过，以前只知道项目中用到过，具体说还真不知道该怎么说。于是，我花了点功夫总结了一下，也做了个例子，拿出来和读者分享一下。实例中主要用到的是对值栈中的 Action 实例和 Struts 2 中的命名对象的访问与操作。

7.4.3 实例应用

【例 7-4】 获取建材信息。

(1) 首先封装一个建筑材料实体类 Material，声明材料名(materialName)、材料价格(mainbid)和材料数量(mount)属性，并分别定义所有属性的 get 和 set 方法，代码如下所示。

```
public class Material {
    private String materialName;    //材料名
    private int mainbid;    //材料价格
    private int mount;  //材料数量
    //构造初始化数据
    public Material(String materialName, int mainbid, int mount) {//构造函数
```

```
        super();
        this.materialName = materialName;
        this.mainbid = mainbid;
        this.mount = mount;
    }

    public String getMaterialName(){
        return materialName;
    }

    public void setMaterialName(String materialName) {
        this.materialName = materialName;
    }
    //下面是属性材料价格 mainbid 和材料数量 mount 的 get、set 方法，在此省略了。
}
```

(2) 然后定义一个 AddOgnlAction，声明 Struts 2 中的两个命名对象 request 和 session 对象，还声明一个存储材料对象的集合 materials，接下来重写继承自 ActionSupport 类的 execute()方法，在该方法中向 request、session 对象中放入 materialName 参数，然后向材料集合中添加材料对象，代码如下所示。

```
public class AddOgnlAction extends ActionSupport {
    //设置 request、response 参数和需要显示的数据集合定义
    private HttpServletRequest request;
    private HttpSession session;
    private List<Material> materials;
    public String execute() throws Exception {
        request = ServletActionContext.getRequest();
        session=request.getSession();
        //设置 request、session 存放值
        request.setAttribute("materialName","人造石台面 From request");
        session.setAttribute("materialName","欧龙无苯油漆(六度) From session");

        //初始化数据集合，集合类型为 List
        materials = new ArrayList<Material>();
        materials .add(new Material("欧龙无苯油漆(六度)", 100, 2000));
        materials .add(new Material("6*10mm 门套线红影木夹板饰面(单面)", 20,
2900));
        materials .add(new Material("人造石台面", 56, 800));
        return SUCCESS;
    }
    public HttpServletRequest getRequest() {
        return request;
    }
    public void setRequest(HttpServletRequest request) {
        this.request = request;
```

```
    }
    public HttpSession getSession() {
        return session;
    }
    public void setSession(HttpSession session) {
        this.session = session;
    }
    public List<Material> getMaterials() {
        return materials;
    }
    public void setMaterials(List<Material> materials) {
        this.materials = materials;
    }
}
```

(3) 接下来打开项目 src 目录下的 struts.xml 文件，向该文件中添加 AddOgnlAction 的配置，代码如下所示。

```
<package name="AddOgnl" extends="struts-default">
    <!-- 创建 Action -->
    <action name="add_ognl" class="struts2.action.AddOgnlAction">
            <result name="success">/add_ognl.jsp</result>
    </action>
</package>
```

(4) 最后新建一个 add_ognl.jsp 文件，在该文件中使用 Struts 2 中的命名对象 request 和 session 直接获取它们中的参数，直接访问栈顶 Action 实例的材料列表属性 materials，输出显示材料信息，代码如下所示。

```
<body>
    <!--  OGNL 显示 request、response 中的值  -->
    <h3 align="left">Session 和 Request 值</h3>
    request.materialName: <s:property value="#request.materialName" /><br/>
    session.materialName: <s:property value="#session.get('materialName')"
/><br/>

    <!--  OGNL 显示条件表达式过滤的数据  -->
    <h3 align="left">根据条件显示数据</h3>
    <p>价格小于 50 元的建材</p>
    <ul>
        <s:iterator value="materials.{?#this.mainbid <50}">
            <li><s:property value="materialName"/> 建材价格是 <s:property
value="mainbid"/>
            元!</li>
        </s:iterator>
    </ul>
    <p> "人造石台面" 的库存数量是: <s:property
value="materials.{?#this.materialName=='人造
```

```
石台面'}.{mount}[0]"/></p>
    <!-- OGNL 新建 Map 类型数据集合，显示子元素值 -->
    <h3 align="left">Map 数据显示</h3>
    <s:set name="frank" value="#{'material':'欧龙无苯油漆(六度)',
'mount':'500'}" />
    <p>供销商 frank 手里还有建材 <s:property value="#frank['material']" /></p>
    <p>库存量为<s:property value="#frank['mount']" /></p>
</body>
```

7.4.4　运行结果

打开 IE 浏览器，在地址栏中输入“http://localhost:8080/Struts2_7/add_ognl.action”，执行效果如图 7-8 所示。



图 7-8　获取建材信息

7.4.5　实例分析



源码解析：

上述案例中，我们封装了一个建筑材料实体类 Material，并创建了一个 AddOgnlAction，在该 Action 中声明了 Struts 2 中的命名对象 request 和 session 对象和一个材料对象集合 materials。当用户请求 AddOgnlAction 成功时，跳转到 add_ognl.jsp 页面中。由于 request 对象和 session 对象不在值栈中，所以在 ActionContext 中，使用加上“#”前缀的表达式来访问它们中的参数，如项目中的：“#request.materialName”。由于 Action 实例默认放在值栈的栈顶，所以可以不加“#”前缀，直接访问 AddOgnlAction 中的材料对象集合 materials，如项目中的：“materials.{?#this.mainbid <50}”。

7.5 常见问题解答

7.5.1 OGNL运算问题



OGNL 运算问题?
网络课堂：http://bbs.itzcn.com/thread-10996-1-1.html

request 作用域里面有一变量 cur=4，在页面中用 struts2 的 if 标签将它取出，代码如下。

```
<s: if test="#request.cur==4"></s:if>
```

这样只能实现简单的逻辑，现在想把 cur 除以 5，看余数是否为 1。求高手解答。

【解决办法】：这个不难实现，其代码如下所示。

```
<s:set name="cur" scope="request" value="4"></s:set>
<s:if test="(#request.cur%5)==1">
    取余为 1
</s:if>
<s:else>
    取余不为 1
</s:else>
```

7.5.2 OGNL调用方法：#session.cart.showcart()访问不到



ognl 调用方法：#session.cart.showcart()访问不到?
网络课堂：http://bbs.itzcn.com/thread-11002-1-1.html

我用的是 Struts 2，一个类放在 session 中，想用 ognl 调用这个方法：#session.cart. showcart()，但访问不到，不知道是怎么回事?

给出错误 Warning，如下所示。

```
[com.opensymphony.xwork2.ognl.OgnlValueStack] Could not find method
[#session.cart.showcart()]
```

从错误中可以看出，session 里有这个类{cart=com.shoppingcart.service.impl. Cart@1999b96}但就是访问不到这个 Cart 里面的方法。请高手帮忙解答。

【解决办法】：这个错误意思是，在 Servlet 中运行 service 方法调用 method 时，找不到所调用的 method(#session.cart.showcart())，你这个方法是个内部方法，在外面调用不到！因此会报错！也有可能是方法名大小写问题。

7.5.3　后台报错：Caught OgnlException while setting property 'operateResult' on type怎么回事

后台报错：Caught OgnlException while setting property 'operateResult' on type 是怎么回事？

网络课堂：http://bbs.itzcn.com/thread-11004-1-1.html

遇到了一个很棘手的 OGNL 异常问题，后台报错输出如下。

```
Caught OgnlException while setting property 'operateResult' on type
'org.apache.struts2.dispatcher.ServletActionRedirectResult'.
ognl.NoSuchPropertyException:
org.apache.struts2.dispatcher.ServletActionRedirectResult.operateResult
at ognl.ObjectPropertyAccessor.setProperty(ObjectPropertyAccessor.java:132)
at
com.opensymphony.xwork2.util.OgnlValueStack$ObjectAccessor.setProperty(Ognl
ValueStack.java:82)
at ognl.OgnlRuntime.setProperty(OgnlRuntime.java:1656)
at ognl.ASTProperty.setValueBody(ASTProperty.java:101)
at ognl.SimpleNode.evaluateSetValueBody(SimpleNode.java:177)
at ognl.SimpleNode.setValue(SimpleNode.java:246)
at ognl.Ognl.setValue(Ognl.java:476)
at com.opensymphony.xwork2.util.OgnlUtil.setValue(OgnlUtil.java:186)
at
com.opensymphony.xwork2.util.OgnlUtil.internalSetProperty(OgnlUtil.java:360)
at com.opensymphony.xwork2.util.OgnlUtil.setProperties(OgnlUtil.java:76)
at
com.opensymphony.xwork2.ObjectFactory.buildResult(ObjectFactory.java:222)
at
com.opensymphony.xwork2.DefaultActionInvocation.createResult(DefaultActionI
nvocation.java:195)
```

下面是 struts.xml 配置文件里的内容。

```
<action name="UnlockEnterprise"

class="com.baosight.worksheetmanagement.action.WorksheetEnterpriseUnlockAct
ion">
    <result name="success">/jsp/success.jsp</result>
    <result name="fail">/jsp/error.jsp</result>
    <result name="worksheetUnlockAction" type="redirectAction">
    <param name="actionName">worksheetUnlock</param>
    <param name="operateResult">${operateResult}</param>
    <param name="operate">success</param>
    </result>
</action>
```

上面配置的 operateResult 有相应的 get、set 方法。请问此问题如何解决，产生的原因是什么。

【解决办法】：你用的 ResultType 是 redirectAction，这是进行浏览器的重定向，并且重定向到的是一个 Action。

当 UnlockEnterprise 的 Action 返回的“worksheetUnlockAction”的视图时，客户端的浏览器就会重新发出一个请求，即在地址栏中可以看到。

```
xxxx.action?operateResult=YYYYY&operate=success;
```

其中的 XXXX 就是 UnlockEnterprise 的返回的视图中的。

```
<param name="actionName">worksheetUnlock</param>
```

YYYYY 如下所示。

```
<param name="operateResult">${operateResult}</param>
```

应该将 type 配成"chain"。代码没问题，此问题属于配置文件的问题，仔细检查配置文件有没有配置出错，或者换个 ognl 包。ognl.NoSuchPropertyException:org.apache.struts2.dispatcher.ServletActionRedirectResult.operateResult，也可能是 Struts 2 的问题。

7.5.4 JSP脚本在Struts 2 中利用OGNL和标签如何表示



JSP 脚本在 Struts 2 中利用 OGNL 和标签如何表示？
网络课堂：http://bbs.itzcn.com/thread-11008-1-1.html

本人初学 Struts 2，对 OGNL 也是一无所知，听说 OGNL 表达式功能强大，所以想问一下如果有如下一句 JSP 脚本。

```
<%=prs.getToolBar("/zsw/user.do?method=getlist")%>
```

利用 Struts 2 中的 OGNL 和标签如何表示？

【解决办法】：你的 prs 是什么？

如果是 Action 中的方法的话，那么在 Struts 2 标签中可以直接使用，例如下面所示。

```
<s:property value="getToolBar('....')"/>
```

如果是某类的静态方法，可以这么调用。

```
<s:property value="@类名(完整)@方法名(或者属性名)"/>
```

以上是 OGNL 表达式对方法的调用。

7.6 习　　题

一、填空题

(1) OGNL 三要素：表达式(Exception)、__________和上下文环境(Context)。

(2) OGNL 支持的常量类型有：字符串常量、字符常量、__________、布尔常量和 null 常量。

(3) OGNL 表达式中的 in 和 not in 操作符的作用：__________。

(4) OGNL 有一个简化的 lambda 表达式语法，能够让你写一些简单的功能。定义 lambda 表达式的语法是__________。

二、选择题

(1) 假如有一个 employee 对象作为 OGNL 上下文的根对象，那么访问 employee 对象的 name 属性，下列表达式哪个正确？__________

A. name　　　　B. #employee.name

C. #name　　　　D. employee.getName()

(2) 下列对象哪些是 Struts 2 中的命名对象？__________

A. page　　　　B. exception

C. parameter　　　　D. application

(3) Struts 2 在 OGNL 基础上的增强有哪些？__________

A. 值栈(ValueStack)　　　　B. EL 表达式

C. N 语法　　　　D. lambda 表达式

(4) 使用 lambda 表达式写一个斐波那契函数，下面哪个正确？__________

A. fib=:[#this==0?0:#this==1?1:#fib(#this-2)+fib(#this-1)]

B. #fib=:[this==0?0:this==1?1:fib(this-2)+fib(this-1)]

C. #fib=:[#this==0?0:#this==1?1:#fib(#this-2)+#fib(#this-1)]

D. fib=:[this==0?0:this==1?1:fib(this-2)+fib(this-1)]

三、上机练习

上机练习：筛选出班里成年的学生。

本次上机练习要求：首先封装一个 Student 学生实体类，包含两个属性 name 和 age。然后创建一个 StudentAction，在该 Action 中声明一个 List 列表存储 Struts 对象。最后新建一个 student.jsp 页面在该页面中应用 OGNL 对集合操作的知识，依次筛选出班里成年学生的信息，显示在页面上。

最终运行后，执行输出结果如图 7-9 所示。



图 7-9　成年学生的信息



第 8 章　Struts 2 的标签库

内容摘要：

一个 MVC 框架，重点是实现两部分：控制器部分和视图部分。Struts 2 框架同样也把重点放在了这两个部分：控制器部分是由 Action(以及隐藏的系列拦截器)来提供支持的，视图部分则通过大量的标签来提供支持。

Struts 2 提供的标签库功能非常强大，而且非常好用。使用标签来开发，可以使页面更加整洁容易维护，减少代码量以及开发时间。本章介绍了标签库的用法，包括如何通过标签库来改进 JSP 页面的数据显示，和使用主题、模板支持，如何简化视图页面的编写等。下面将详细地讲解 Struts 2 标签库各方面的使用。

学习目标：

- 掌握标签的语法。
- 熟练数据标签和控制标签的使用。
- 掌握模板和主题。
- 熟练表单标签的使用。
- 理解非表单标签。

8.1 演员年龄的排序

Struts 2 提供的非 UI 标签包括控制标签和数据标签，主要用于完成流程控制和对 ValueStack 的控制。数据标签主要用于访问 ValueStack 中的数据，控制标签可以完成输出流程控制。本节将详细讲解控制标签的应用。

视频教学： 光盘/videos/08/iterator.avi　　长度：6 分钟
光盘/videos/08/append.avi　　长度：7 分钟
光盘/videos/08/generator.avi　　长度：6 分钟
光盘/videos/08/ subset.avi　　长度：9 分钟
光盘/videos/08/ sort.avi　　长度：5 分钟

8.1.1 基础知识——控制标签

控制标签可以完成输出流程控制，例如分支、循环等操作；也可完成对集合的合并、排序等操作。控制标签有：if、elseIf/elseif、else、append、generator，等等，关于它们的讲解如下。

1. if/elseif/else标签

if/elseif/else 这三个标签都是用于分支控制的，它们都用于根据一个 Boolean 表达式的值，来决定是否计算、输出标签体的内容。

这三个标签可以结合使用，<s:if .../>标签可以单独使用，但<s:elseif .../>和<s:else .../>都不可单独使用，必须与<s:if .../>结合使用。<s:if .../>标签可以与多个<s:elseif .../>标签结合使用，并可以结合一个<s:else .../>标签使用。三个标签结合使用的语法格式如下所示。

```
<s:if test="表达式">
    标签体
</s:if>
<s:elseif test="表达式">
    标签体
</s:elseif>
<!-- 允许出现多次 elseif 标签 -->
…
<s:else>
    标签体
</s:else>
```

上面的 if/elseif/else 三个标签组合，对应了 Java 语言里的 if{}else{}分支结构。

2. iterator标签

iterator 标签主要用于集合的迭代，这里的集合包含 List、Set 和数组，也可对 Map 类型的

对象进行迭代输出。

使用<s:iterator.../>标签对集合进行迭代输出时，可以指定如表 8-1 中的三个属性。

表 8-1　iterator标签属性

属性名称	是否必需	默认值	类　型	说　明
value	否	无	String	指定被迭代的集合，如果没有指定 value 属性，则使用 ValueStack 栈顶的集合
id	否	无	Collection、Map、Enumeration、Iterator 或者 Array	指定集合里元素的 id
status	否	无	String	指定迭代时的 IteratorStatus 实例，通过该实例即可判断当前迭代元素的属性。例如是否是最后一个，以及当前迭代元素的索引等



如果为 status 属性指定一个值，将创建一个 org.apache.struts2.views.jsp.IteratorStatus 实例。

IteratorStatus 类中有如下的方法。

- public int getCount()：获取当前迭代的元素的总数。
- public int getIndex()：获取当前迭代的元素的索引。
- public boolean isEven()：判断当前迭代的元素的顺序是否是偶数。
- public boolean isOdd()：判断当前迭代的元素的顺序是否是奇数。
- public boolean isFirst()：判断当前迭代的元素是否是第一个元素。
- public boolean isLast()：判断当前迭代的元素是否是最后一个元素。

通过上面几个方法，可以在迭代时，根据当前迭代元素的属性，来进行更多的控制，下面通过这些属性和 iterator 标签来迭代显示节日集合，代码如下所示。

```
<table border="0" >
    <s:iterator value="{'端午节','中秋节','七夕节','重阳节'}" id="name"
status="status">
        <tr <s:if test="#status.odd">style="background-color:red"</s:if>>
            <td><s:property value="#status.count"/><s:property
value="name"/></td>
        </tr>
    </s:iterator>
</table>
```

3. append标签

append 标签可以将多个集合对象拼接起来，组成一个新的集合。通过这种拼接，允许通过一个<s:iterator .../>标签就完成对多个集合的迭代。

append 标签可以指定一个 id 属性，如果指定了该属性，那么组合后的迭代器将被保存到 OgnlContext 中，可以通过该属性的值来引用组合后的迭代器。

当使用<s:append …/>标签时，需要指定一个 id 属性，该属性确定拼接生成的新集合的名字。除此之外，<s:append …/>标签还可以接受多个<s:param …/>子标签，每个子标签指定一个集合，<s:append …/>标签负责将<s:param …/>标签指定的多个集合拼接成一个集合。

下面使用 append 标签将两个集合拼接成一个新集合，然后使用 iterator 标签对新集合进行迭代。例子如下代码。

```
<!-- 使用 append 标签将两个集合拼接成新的集合，新集合的名字是 day -->
<s:append id="day">
    <s:param value="{'端午节','中秋节','七夕节','重阳节'}" />
    <s:param value="{'元旦','春节'}" />
</s:append>
<table border="0" >
    <s:iterator value="#day" status="status">
    <!-- 对新集合进行迭代，使用 status 属性-->
        <tr <s:if test="#status.odd">style="background-color:red"</s:if>>
            <td><s:property/></td>
        </tr>
    </s:iterator>
</table>
```

4．merge标签

merge 标签的用法看起来与 append 标签非常像，它也是用于将多个集合拼接成一个新集合。它们的区别就是对合并后的迭代器中的元素迭代的顺序不一样。

假如有三个迭代器被合并，每一个迭代器有三个元素，下面是使用 merge 标签合并后的迭代器中的元素被迭代的顺序，如图 8-1 所示。

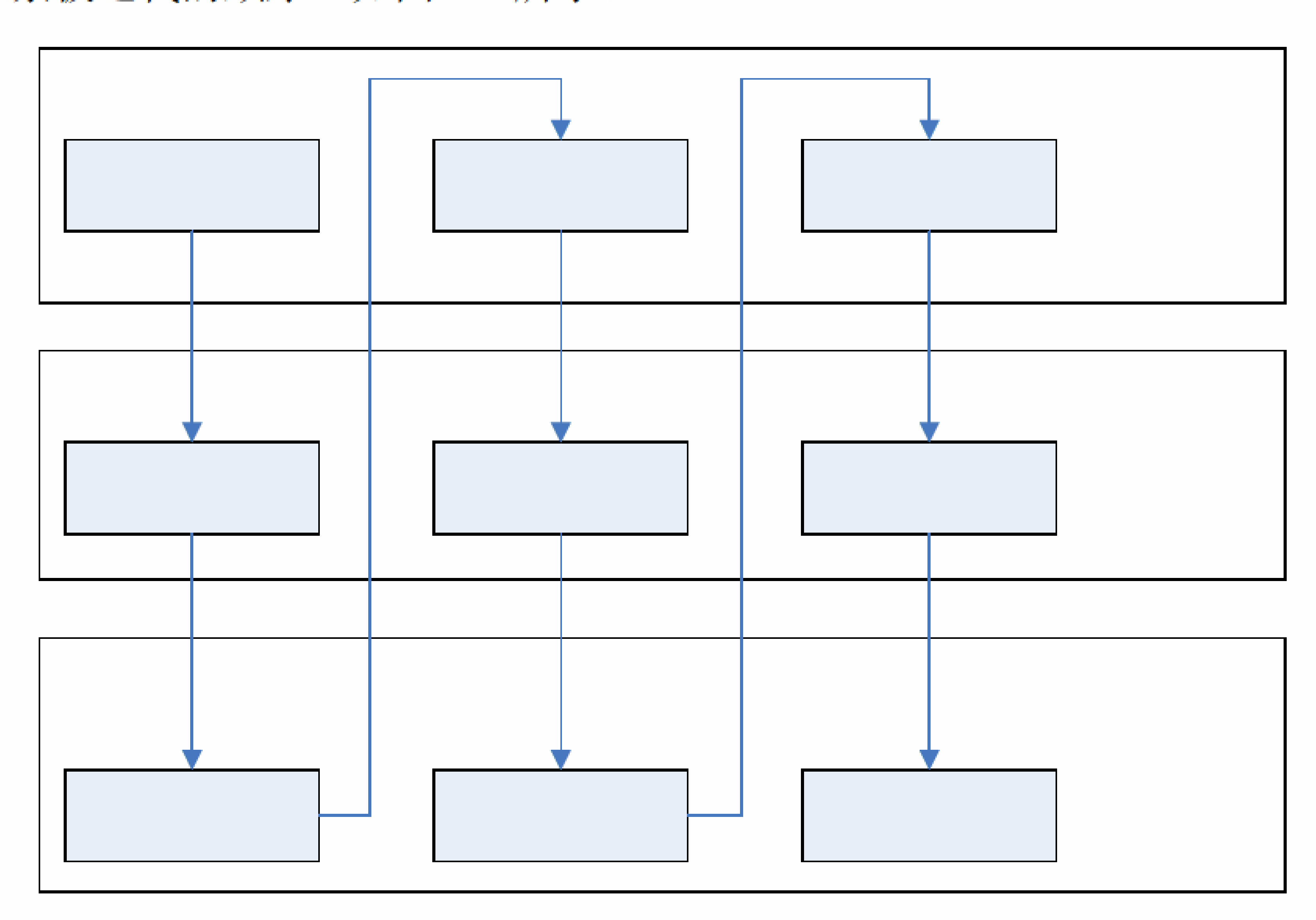


图 8-1　merge标签合并后的迭代器的元素迭代顺序

下面是使用 append 标签合并后的迭代器中的元素被迭代的顺序，如图 8-2 所示。

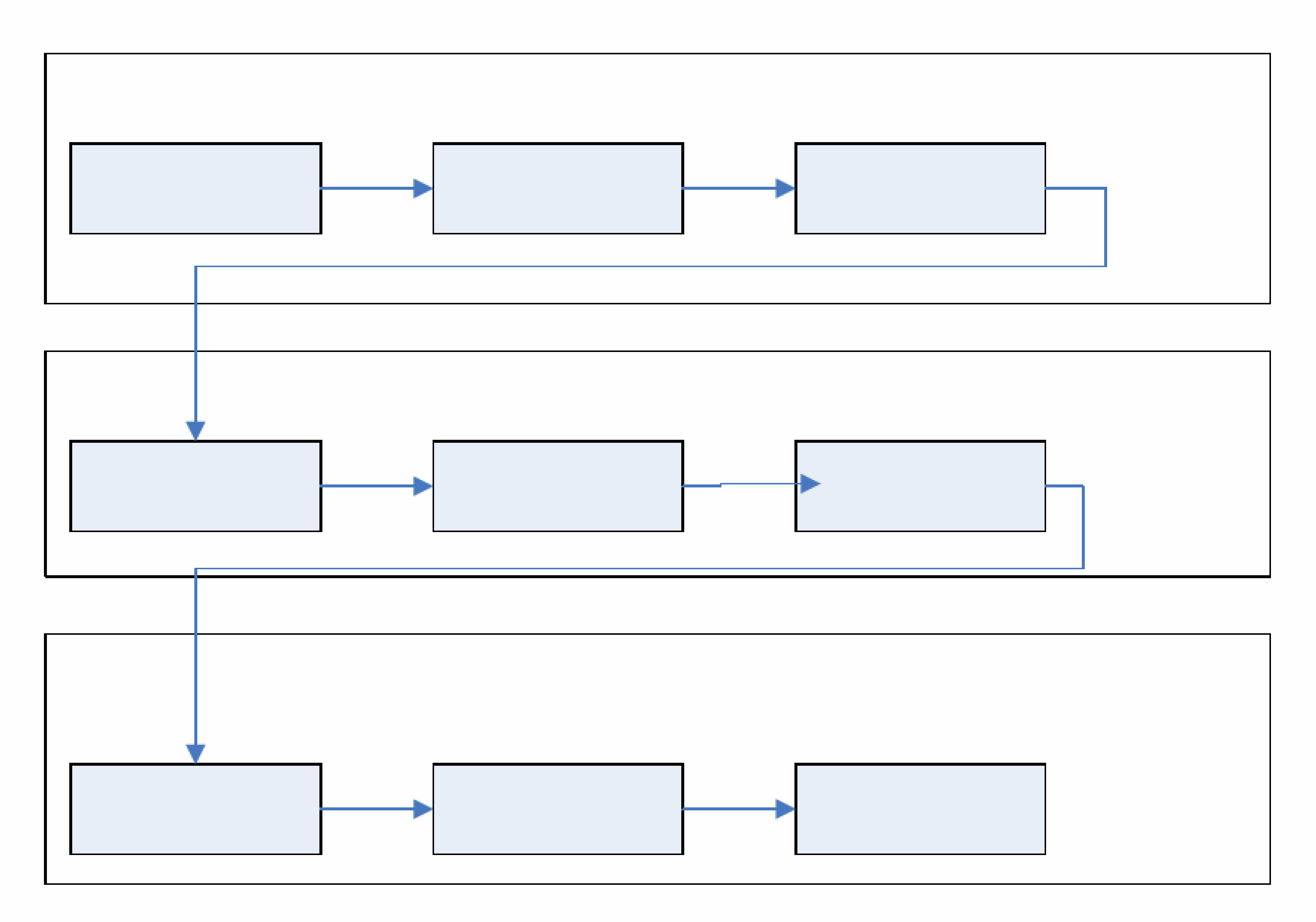


图 8-2　append标签合并后的迭代器中元素被迭代的顺序

merge 标签有一个 id 属性，如果指定了该属性那么合并后的迭代器将被保存到 OgnlContext 中，可以通过该属性的值来引用合并后的迭代器。

第一个元素

通过一个小程序，看看 merge 标签是如何来拼接多个集合的，代码如下所示。

```
<s:merge id="mergeday">
    <s:param value="{'元宵节','清明节'}"></s:param>
    <s:param value="{'端午节','中秋节','七夕节','重阳节'}" />
    <s:param value="{'元旦','春节'}" />
</s:merge>
<table border="0" >
    <s:iterator value="#mergeday" status="status">
        <tr <s:if test="#status.odd">style="background-color:red"</s:if>>
            <td><s:property/></td>
        </tr>
    </s:iterator>
</table>
```

上述代码使用 merge 标签，将三个节日集合拼接成为一个新集合，并通过 iterator 标签依次显示在表格中。

第一个元素

5．generator标签

generator 标签根据 separator 属性指定的分隔符，将 val 属性指定的值进行拆分，然后生成一个迭代器，压入到值栈的栈顶。在 generator 标签的内部，可以使用 iterator 标签取出栈顶的迭代器对拆分后的各个部分进行迭代。当 generator 标签结束时，栈顶的迭代器将被删除。也可以这样理解：generator 将一个字符串转化成一个集合。在该标签的内部，整个临时生成的集合将位于 ValueStack 的顶端，一旦该标签结束，该集合将被移出 ValueStack。

generator 标签的属性如表 8-2 所示。

表 8-2　generator标签的属性

属性名称	是否必需	默 认 值	类　型	说　明
val	是	无	String	指定要解析的值
separator	是	无	String	指定用于解析 val 属性的分隔符
count	否	无	Integer	指定在生成的迭代器中可用的元素数量
converter	否	无	org.apache.struts2.util.IteratorGenerator.Converter	指定一个转换器，用于将解析后的各个字符串转换为对象
id	否	无	String	如果指定了该属性，那么生成的迭代器将以该属性的值为 key 保存到 pageContext 对象中

下面的代码使用 generator 标签生成一个简单的集合。详细代码如下所示。

```
<table border="0" >
    <!-- 使用 s:generator 标签将一个字符串解析转换成一个集合 -->
    <s:generator val="'端午节,中秋节,七夕节,重阳节'" separator=",">
        <!-- 在 generator 标签内，该集合位于 ValueStack 的栈顶，故此处迭代就是临时生成
的集合
        -->
        <s:iterator status="status">
            <!-- 根据当前迭代项索引的奇偶来决定是否使用 css 样式 -->
            <tr <s:if
test="#status.odd">style="background-color:red"</s:if>>
                <td><s:property /></td>
            </tr>
        </s:iterator>
    </s:generator>
</table>
```

6．subset标签

subset 标签用于取得集合的子集，该标签的底层通过 org.apache.Struts2.util.SubsetIteratorFilter 类提供实现。

使用 subset 标签时，可以指定的属性如表 8-3 所示。

表 8-3　subset标签属性

属性名称	是否必需	默 认 值	类　型	说　明
count	否	无	Integer	指定子集中元素的个数。如果不指定该属性，默认取得源集合的全部元素
source	否	无	Collection、Map、Enumeration、Iterator 或者 Array	指定源集合。如果不指定该属性，默认取得 ValueStack 栈顶的集合
start	否	0	Integer	指定子集从源集合的第几个元素开始截取。默认从第一个元素(即 start 的默认值为 0)开始截取
decider	否	无	org.apache.struts2.util.SubsetIteratorFilter.Decider	指定有开发者自己决定是否选中该元素

下面向读者讲解一个小案例。在这个案例中，使用 subset 元素先指定 source 属性源集合，然后指定 start 属性为 2，即标签子集从源集合的第三个元素开始截取，最后指定 count 属性为 4，表明自己的长度为 4。代码如下所示。

```
<table border="0" >
    <!-- 使用 s:subset 标签截取目标集合的四个元素，从第二个元素开始截取-->
    <s:subset source="{'春节','元宵节','清明节','端午节','五一劳动节','中秋节'}"
start="2" count="4">
        <!-- 使用 iterator 标签来迭代目标集合，因为没有指定 value 属性值，故迭代
ValueStack 栈
        顶的元素 -->
        <s:iterator status="status">
            <!-- 根据当前迭代项索引的奇偶来决定是否使用 CSS 样式 -->
            <tr <s:if
test="#status.odd">style="background-color:red"</s:if>>
                <td><s:property /></td>
            </tr>
        </s:iterator>
    </s:subset>
</table>
```



在 subset 标签内部时，subset 标签生成的子集放在 ValueStack 的栈顶，如果该标签结束后，该标签生成的子集将被移出 ValueStack 栈。

7．sort标签

sort 标签用于对指定的集合元素进行排序。进行排序时，必须提供自身的排序规则，即使实现自己的 Comparator，Comparator 也需要实现 java.util.Comparator 接口。

使用 sort 标签时，可以指定如表 8-4 中的属性。

表 8-4　sort标签属性

属性名称	是否必需	默 认 值	类　型	说　明
comparator	是	无	java.util.Comparator	指定进行排序的 Comparator 实例
source	否	无	Collection 、 Map 、 Enumeration、Iterator 或者 Array	指定被排序的集合。如果不指定该属性，则对 ValueStack 栈顶的集合进行排序

下面封装一个 Comparator，代码如下所示。

```
public class SortComparator implements Comparator{
    public int compare(Object e1, Object e2) {
        String str1 = (String)e1;
        String str2 = (String)e2;
        return str1.compareTo(str2);
    }
}
```

实现比较器 SortComparator，需要实现一个 Comparator 接口，并实现它的 compare(Object args1,Object args2)方法。如果该方法返回一个大于 0 的整数，则第一个元素大于第二个元素；如果该方法返回 0，则两个元素相等；如果该方法返回小于 0 的整数，则第一个元素小于第二个元素。

下面代码使用<s:bean>标签引入比较器 SortComparator，然后使用<s:sort>标签将指定的源集合进行排序。代码如下所示。

```
<!-- 使用 bean 标签定义一个 Comparator 实例 -->
<s:bean name="com.struts2.tags.SortComparator" id="sortComparator"></s:bean>
<table border="0">
    <!-- 使用自定义的排序规则对目标集合进行排序 -->
    <s:sort source="{'端午节','中秋节','七夕节','重阳节'}"
comparator="#sortComparator">
        <!-- 迭代输出集合 -->
        <s:iterator>
            <tr><td><s:property/></td></tr>
        </s:iterator>
    </s:sort>
</table>
```

8.1.2 实例描述

之前如果想在 JSP 页面中动态控制页面的输出信息，需要在页面中添加 JSP 脚本，但这样使页面中分布了很多 Java 代码，使人看起来很凌乱。Struts 2 的标签库提供了控制标签，使用这些控制标签可以灵活地控制页面的输出内容。本节实例为读者显现的是，如何使用控制标签实现演员的年龄排序。

8.1.3 实例应用

【例 8-1】 演员年龄的排序。

(1) 在项目目录 src/com.struts2.domain 下，新建一个 Actor 类，声明三个属性 name、sex 和 age。并分别给出这三个属性的 get、set 方法。代码如下所示。

```
public class Actor {
    private String name;        //名称
    private String sex;     //性别
    private int age;            //年龄

    //下面是 name、sex 和 age 属性的 get 和 set 方法，在此省略了。
    //…………………………………………………….
}
```

(2) 在项目目录 src/com.struts2.action 下新建一个 ActorAction 类，声明一个演员列表，并重写继承于 ActionSupport 类的 execute()方法，代码如下所示。

```
public class ActorAction extends ActionSupport{
    private List<Actor> actors = new ArrayList<Actor>();   //演员列表对象

    public List<Actor> getActors() {
        return actors;
    }

    public void setActors(List<Actor> actors) {
        this.actors = actors;
    }

    public String execute() throws Exception{
        //下面向演员列表 actors 中添加演员对象
        actors.add(new Actor("李璐璐",26));
        actors.add(new Actor("郭晓静",32));
        actors.add(new Actor("刘晶晶",35));
        actors.add(new Actor("肖庆宇",38));
        actors.add(new Actor("周丽丽",21));
        actors.add(new Actor("武小倩",13));
        return "success";
    }
}
```

(3) 打开项目目录 src 下的 struts.xml 文件，在该文件中添加 ActorAction 的配置，代码如下所示。

```
<package name="controltag" extends="struts-default">
    <action name="actor" class="com.struts2.action.ActorAction">
        <result>/actor.jsp</result>
    </action>
</package>
```

(4) 在项目目录 src/comparator 下新建一个比较演员年龄的比较器类 SortComparator.java，该类实现 Comparator 接口，并实现该接口的 compare()方法，对演员的年龄进行比较，代码如下所示。

```
public class SortComparator implements Comparator{
    public int compare(Object e1, Object e2) { //比较演员对象
        Actor actor1= (Actor)e1;    //转换成 Actor 对象
        Actor actor2= (Actor)e2;
        Integer int1 = actor1.getAge();//获取演员对象的年龄
        Integer int2 = actor2.getAge();
        return int1.compareTo(int2);       //比较两个演员对象的年龄大小
    }
}
```

(5) 新建一个 actor.jsp 页面，在该页面中使用 iterator 标签显示演员信息。然后使用<bean>标签导入比较器 SortComparator 类实例。最后使用<sort>标签比较年龄大小，并按年龄从小到大排序显示。代码如下所示。

```
<h3><span>演员列表如下：</span></h3>
<p>
<table>
    <s:iterator value="actors" id="name" status="status">
        <tr <s:if test="#status.odd">style="background-color:yellow"</s:if>>
            <td><s:property value="#status.count"/><s:property
value="name"/></td>
        </tr>
    </s:iterator>
</table>
</p>
<h3><span>演员年龄排序如下：</span></h3>
<p>
<!-- 使用 bean 标签定义一个 Comparator 实例 -->
<s:bean name="comparator.SortComparator" id="sortComparator"></s:bean>
<table border="0">
    <!-- 使用自定义的排序规则对目标集合进行排序 -->
    <s:sort source="actors" comparator="#sortComparator">
        <!-- 迭代输出集合 -->
        <s:iterator>
            <tr>
                <td><s:property value="name"/></td>
                <td>年龄：<s:property value="age"/></td>
            </tr>
        </s:iterator>
    </s:sort>
    </table>
</p>
```

8.1.4　运行结果

打开 IE 浏览器，在地址栏中输入“http://localhost:8080/Struts2_8/actor.action”，执行效果如图 8-3 所示。



图 8-3　演员年龄排序

8.1.5 实例分析



源码解析：

上述实例中，首先封装了一个演员实体类 Actor，包含三个属性 name、sex 和 age。

然后创建了一个 ActorAction，在该 Action 中声明一个演员列表对象，并向该列表对象中添加了 Actor 对象。

最后创建了一个演员年龄比较器类 SortComparator 和一个 actor.jsp 页面，在 actor.jsp 页面中使用<bean>标签导入 SortComparator 类实例，并使用<sort>标签将演员年龄按从小到大排序显示。

8.2 显示学员信息

上节详细讲解了控制标签的使用，相信读者对控制执行流程(分支、循环等操作)应该应用自如了。本节将学习如何使用数据标签访问值栈中的数据。

视频教学： 光盘/videos/08/property.avi 长度：5 分钟
光盘/videos/08/set.avii 长度：6 分钟
光盘/videos/08/push.avi 长度：5 分钟
光盘/videos/08/param.avi 长度：5 分钟
光盘/videos/08/bean.avi 长度：5 分钟
光盘/videos/08/action.avi 长度：9 分钟
光盘/videos/08/include.avi 长度：4 分钟
光盘/videos/08/url.avi 长度：8 分钟
光盘/videos/08/date.avi 长度：7 分钟

8.2.1 基础知识——数据标签

数据标签用于访问 ActionContext 和值栈中的数据。数据标签有：property、set、push、param，等等，它们的功能简介以及使用方法如下。

1．property标签

property 标签的作用是输出 value 属性指定的值，如果没有指定 value 属性，则默认输出 ValueStack 栈顶的值。使用该标签可以指定如表 8-5 所示的属性。

property 标签用法比较简单，在上节的实例中已经大量使用到了它，在此来看一个小例子。

```
<s:property value="day" default="七夕节"/>
```

取出栈顶对象(通常是 Action)的 day 属性并输出，如果没有找到 day 属性，则输出“七夕节”。

表 8-5　property标签属性

属性名称	是否必需	默 认 值	类　型	说　明
default	否	无	String	如果需要输出的属性值为 null，则显示的default 属性指定的值
escape	否	true	Boolean	指定是否 escape HTML 代码
value	否	栈顶对象	Object	指定需要输出的属性值，如果没有指定该属性，则默认输出 ValueStack 栈顶的值
id	否	无	String	指定该元素的标识

2．set标签

set 标签用于将某个值放入指定范围内，例如 application 范围、session 范围等。set 标签在某些情况下是比较有用的，例如，在页面中多次引用一个复杂的表达式，并将这个表达式赋给一个变量，然后直接引用变量。使用 set 标签的好处有两个。

(1) 提高了代码的可读性。

(2) 提升了性能(表达式的计算只有一次)。

set 标签的属性如表 8-6 所示。

表 8-6　set标签属性

属性名称	是否必需	默 认 值	类　型	说　明
name	是	无	String	重新生成的新变量的名字
value	否	栈顶对象	Object	指定将赋给变量的值。如果没有指定该属性，则将 ValueStack 栈顶的值赋给新变量
scope	否	Action	String	指定新变量被放置的范围，该属性可以接受application
id	否	无	String	指定该元素的引用 ID



使用 set 标签可以理解为定义一个新变量，且将一个已有的值复制给新变量，并且可以将新变量放到指定的范围内。

set 标签以 name 属性的值作为键(key)，将 value 属性的值保存到指定的范围对象中(如 page、request 和 session 等)。属性 scope 取值中的 page、request、session 和 application 同 JSP 的 4 种范围，如果指定 action 范围(默认值)，value 属性的值将被同时保存到 request 范围和 OgnlContext 中。下面使用 set 标签，将节日对象放到指定的范围内，如：request、session 和 application 等，并将它们取出显示在页面上，代码如下所示。

```
<s:bean name="com.struts2.domain.Festival" id="fest">
    <s:param name="name" value="'端午节'"/>
    <s:param name="desc" value="'为了纪念屈原，而流传下来的节日'"/>
</s:bean>
<!-- 默认将 fest 保存到 OgnlContext 中 -->
```

```
<s:set value="#fest" name="XXX"/>
<s:property value="#XXX.name"/><br/>
<s:property value="#XXX.desc"/><br/>
<!-- 将 fest 保存到 application 范围内 -->
<s:set value="#fest" name="XXX" scope="application"/>
<s:property value="#attr.XXX.name"/><br/>
<s:property value="#attr.XXX.desc"/><br/>
<!-- 将 fest 保存到 session 范围内 -->
<s:set value="#fest" name="XXX" scope="session"/>
${sessionScope.XXX.name}<br/>
${sessionScope.XXX.desc}<br/>
```

3. push标签

push 标签用于将某个值放到 ValueStack 的栈顶，从而可以更简单地访问该值。使用该标签时可以指定如表 8-7 所示的属性。

表 8-7　push标签属性

属性名称	是否必需	默 认 值	类　型	说　明
id	否	无	String	指定引用该标签的 ID
value	是	无	Object	指定需要放到 ValueStack 栈顶的值



push 标签与 set 标签的区别在于，set 标签是将值放到 Action 上下文中。当 push 标签结束后，push 标签放入值栈中的对象将被删除。换句话说，要访问 push 标签压入栈中的对象，需要在标签内部去访问。

如果对某个对象操作比较频繁，可以使用 push 标签将这个对象压入值栈的顶部，随后针对该对象的操作就可以简化了。创建一个 PushAction，将 Festival 对象放到 session 中，代码如下所示。

```
public class PushAction extends ActionSupport {
    private Festival fest;
    public Festival getFest() {
        return fest;
    }
    public void setFest(Festival fest) {
        this.fest = fest;
    }
    public String execute() throws Exception {
        fest = new Festival();
        fest.setName("重阳节");
        fest.setDesc("九月九日望乡台");
        ActionContext.getContext().getSession().put("fest",fest);
        return "success";
    }
}
```

使用<s:property .../>标签直接来访问 session 中的 fest 对象，也可以使用 push 标签将 session 中的 fest 对象压入栈顶，以便访问，代码如下所示。

```
<!-- 访问 session 中的 fest 对象的属性 -->
节日名称：<s:property value="#session.fest.name"/><br/>
节日描述：<s:property value="#session.fest.desc"/><br/>
<!-- 使用 push 标签将 session 中的 fest 对象放到栈顶，便于访问 -->
<s:push value="#session.fest">
    节日名称：<s:property value="name"/>
    节日描述：<s:property value="desc"/>
</s:push>
```

4. param标签

param 标签主要用于为其他标签提供参数，例如：为 append 标签、merge 标签、bean 标签和 include 标签提供参数。param 标签可以配置如表 8-8 所示的属性。

表 8-8　param标签属性

属性名称	是否必需	默 认 值	类　型	说　明
name	否	无	String	指定需要设置参数的参数名
value	否	无	Object	指定需要设置参数的参数值
id	否	无	String	指定引用该元素的 ID

其中，value 属性是可选的，因为<s:param .../>标签有两种用法。

第一种用法如下所示。

```
<param name="username">Jack</param>
```

在上面的用法中，指定一个名为 username 的参数，该参数的值为 Jack。

第二种用法如下所示。

```
<param name="username" value="Jack"/>
```

在上面的用法中，指定一个名为 username 的参数，该参数的值为 Jack 对象的值——如果 Jack 对象不存在，则 username 参数的值为 null。如果想指定 username 参数的值为 Jack 字符串，则应该写如下代码。

```
<param name="username" value="'Jack'"/>
```



如果采用上面写法，又希望直接传入字符串值，则应该将字符串常量放入引号中。

5. bean标签

bean 标签用于创建一个 JavaBean 实例。创建 JavaBean 实例时，可以在该标签体内使用多个 param 标签来为对象的属性(必须有相应的 setter 方法)注入值。如果 bean 标签还指定了 id 属性，则创建的 JavaBean 对象将被放入 OgnlContext 中。

bean 标签的属性如表 8-9 所示。

表 8-9　bean标签属性

属性名称	是否必需	默 认 值	类　型	说　明
name	是	无	Object	要实例化的 JavaBean 的完整类名
id	否	无	String	指定一个名字，用于引用放入 OgnlContext 中的 JavaBean 对象



Id 属性为一个可选属性，不管是否指定了 id 属性，bean 标签创建的 JavaBean 实例都将被压入 ValueStack 的顶端，在 bean 标签的内部可以直接访问实例化的对象，不用使用“#”标记。但一旦 bean 标签结束了，bean 标签创建的 JavaBean 实例将从 ValueStack 中移除，这样的话就无法访问到该 JavaBean 实例了，除非指定了 id 属性， bean 标签创建的 JavaBean 实例还将被放到 OgnlContext 中，在 bean 标签的外部，依然可以访问创建的对象，不过此时需要要添加“#”标记。

以下是 bean 标签的一个使用练习。这个练习使用 bean 标签来实例化一个节日类 Festival，并演示 bean 标签不指定 id 属性与指定 id 属性的区别。代码如下所示。

```
<h3>bean 标签没有指定 id 属性，创建的 Festival 实例被放到 ValueStack 的顶部</h3>
<s:bean name="com.struts2.domain.Festival">
    <s:param name="name" value="'中秋节'"/>
    <s:param name="desc" value="'中秋节要吃月饼哦'"/>
    节日名称：<s:property value="name"/><br/><!-- 可以输出 name 属性值 -->
    节日描述：<s:property value="desc"/><br/><!-- 可以输出 desc 属性值-->
</s:bean>
<p>
    节日名称：<s:property value="name"/><br/><!-- Festival 对象已从栈顶移除，输出
为 null -->
    节日描述：<s:property value="desc"/><br/><!-- Festival 对象已从栈顶移除，输出
为 null-->
</p>
<h3>bean 标签指定了 id 属性，创建的 Festival 实例被放到 ValueStack 的顶部和 OgnlContext
中</h3>
<s:bean name="com.struts2.domain.Festival" id="fest">
    <s:param name="name" value="'中秋节'"/>
    <s:param name="desc" value="'中秋节要吃月饼哦'" />
    节日名称：<s:property value="name"/><br/><!-- 可以输出 name 属性值 -->
    节日描述：<s:property value="desc"/><br/><!-- 可以输出 desc 属性值 -->
</s:bean>
<p>
    节日名称：<s:property value="#fest.name"/><br/><!-- 可以输出 name 属性值 -->
    节日描述：<s:property value="#fest.desc"/><br/><!-- 可以输出 desc 属性值-->
</p>
```

上述代码中没有指定 id 属性时，创建的 Festival 对象只被压入值栈，在 bean 标签的内部使用 property 标签可以直接访问 Festival 对象的 name 和 desc 属性。在 bean 标签的外部，将无

法访问 Festival 对象的属性，因为此时值栈中的 Festival 对象已被移除。

在指定 id 属性时，创建的 Festival 对象被放到值栈和 Action 上下文中，因此在标签的内部和外都可以访问 Festival 对象的属性，只不过在标签外部访问时需要添加#fest 作为前缀。

6. action标签

通过指定 Action 的名字和可选的名称空间，action 标签允许在 JSP 页面中直接调用 Action。如果将标签的 executeResult 属性设为 true(默认为 false)，那么 Action 对应的结果输出也将被包含到本页面中。



如果为 action 标签指定了 id 属性，则相应的 Action 实例将被放到 OgnlContext 中，在 action 标签结束后，也可以通过#id 来应用 Action。

在 action 标签体中也可以嵌套 param 标签，向 Action 传递参数。action 标签的属性如表 8-10 所示。

表 8-10 action标签的属性

属性名称	是否必需	默 认 值	类 型	说 明
id	否	无	String	该属性将会作为该 Action 的引用 ID
name	是	无	String	指定该标签调用哪个 Action
namespace	否	当前页面所在的名称空间	String	指定该标签调用的 Action 所在的 namespace
executeResult	否	false	Boolean	指定是否要将 Action 的处理结果页面包含到本页面。默认值为 false，即不包含
ignoreContextParams	否	false	Boolean	当 Action 被调用的时候，请求参数是否应该传入 Action
flush	否	true	Boolean	在 action 标签结束时，输出结果是否应该被刷新

下面的实例应用到了一个 Action 类，在 Action 中声明两个属性分别为 festName 和 desc。还有两个方法 execute()和 welcome()。代码如下所示。

```
public class ActionTag extends ActionSupport {
    private String festName;    //节日名称
    private String desc;    //节日描述
    public String getFestName() {
        return festName;
    }
    public String getDesc() {
        return desc;
    }
    public void setDesc(String desc) {
        this.desc = desc;
    }
    public void setFestName(String festName) {
        this.festName = festName;
    }
```

```
    public String execute() throws Exception{
        return "success";
    }
        public String welcome(){
        ServletActionContext.getRequest().setAttribute("welcome", "重阳节陪老
人登高");
        return "success";
    }
}
```

创建一个成功页面 success.jsp，当请求方法 execute()成功之后，将跳转到该页面，代码如下所示。

```
节日名称：<s:property value="festName"/><br/>
节日描述：<s:property value="desc"/><br/>
```

创建一个 welcome_success.jsp 页面，使用<s:action>标签来调用 ActionTag 的两个逻辑方法 execute()和 welcome()。代码如下所示。

```
<h3>执行结果，并将结果页面的输出包含到本页面中</h3>
<s:action name="first" executeResult="true"/>
<h3>不执行结果，调用 ActionTag 的 welcome()方法，获取请求对象中的 welcome 属性</h3>
<s:action name="welcome" executeResult="false"/>
<s:property value="#attr.welcome"/>
<h3>执行结果，并通过嵌套的 param 标签，设置 ActionTag 的 festName 和 desc 属性</h3>
<s:action name="first" executeResult="true">
    <s:param name="festName" value="'元宵节'"></s:param>
    <s:param name="desc" value="'吃元宵的节日哦'"></s:param>
</s:action>
```

上述代码使用 action 标签配置了名为 first 的 Action，并指定了 executeResult 属性来控制是否将处理结果包含到本页面中，请求 Action 成功之后将跳转到 success.jsp 页面，但由于没有为 Action 的 festName 和 desc 属性赋值，所以显示为 null。使用 param 标签为 Action 传入参数之后，将输出显示属性值。使用 action 标签还配置了名为 welcome 的 Action，并使用 property 标签输出显示了 welcome 参数值。

7．include标签

include 标签用于将一个 JSP 页面，或者一个 Servlet 包含到本页面中。它类似于 JSP 中的<jsp:include>标签。在 include 标签体内也可以包含多个 param 标签，向被包含的页面传递请求参数。

include 标签的属性如表 8-11 所示。

表 8-11　include标签的属性

属性名称	是否必需	默 认 值	类　型	说　明
id	否	无	String	指定该标签的 ID 引用
value	是	无	String	指定需要被包含的 JSP 页面或者 Servlet

使用 include 标签来包含 fest1.jsp 和 fest2.jsp 页面，代码如下所示。

```
<h3>使用 include 标签包含 fest1.jsp</h3>
<s:include value="fest1.jsp"/>
<h3>使用 include 标签包含 fest2.jsp，使用 param 标签向 fest2.jsp 传递参数</h3>
<s:include value="fest2.jsp">
    <s:param name="name" value="'七夕节'"/>
    <s:param name="desc" value="'牛郎织女的故事就是从这里开始的'"/>
</s:include>
```

上述代码 include 标签包含了 fest1.jsp 页面，该页面将显示“祝大家七夕节快乐！！！”，使用 include 标签包含 fest2.jsp 页面时，还需要使用 param 标签引入 name 和 desc 参数，在 fest2.jsp 页面中，将使用 EL 表达式获取并显示这两个参数。代码如下所示。

```
祝大家${param.name}快乐！！！${param.desc }
```

8．url标签

url 标签用于生成一个 URL 地址，可以通过为 url 标签指定 param 子元素，从而向指定 URL 发送请求参数。如果 param 标签的 value 属性的值是一个数组或者 Iterator，那么所有的值都会被附加给 URL。

url 标签可以指定的属性如表 8-12 所示。

表 8-12　url标签的属性

属性名称	是否必需	默 认 值	类　型	说　明
Id	否	无	String	指定该 url 元素的引用 ID
includeParams	否	get	String	指定是否包含请求参数，该属性的属性值只能为 none、get 或者 all
scheme	否	无	String	指定 URL 使用的协议(HTTP 或 HTTPS)
action	否	无	String	指定用于生成 URL 的 action，如果没有使用该属性，则使用 value 属性给出的值生成 URL
value	否	无	String	指定用于生成 URL 的地址值，如果没有使用该属性，则使用 Action 属性给出的值生成 URL
anchor	否	无	String	指定 URL 的锚点
encode	否	true	Boolean	指定是否编码生成的 URL，默认值为 true，便于在客户端浏览器不支持 Cookie 时，采用 URL 重写的机制来跟踪 Session
escapeamp	否	true	Boolean	指定是否将“&”号转义为“&amp”
includeContext	否	true	Boolean	指定是否将当前应用程序的上下文路径(context path)包含在生成的 URL 中
method	否	无	String	指定使用的 action 的方法
namespace	否	无	String	指定 action 所属的名称空间
forceAddSchemeHostAndPort	否	无	Boolean	指定是否强制添加 scheme、主机和端口

下面通过 url 标签来配置 Action 请求，并向 Action 中传递参数，代码如下所示。

```
只指定 value 属性的形式。<br>
<s:url value="welcome.action"/>
    指定 action 属性，且使用 param 标签传入参数.<br/>
<s:url action="first">
    <s:param name="festName" value="'元旦'"/>
    <s:param name="desc" value="'1 月 1 日元旦节'"/>
</s:url>
<hr>
既不指定 action 属性，也不指定 value 属性，且使用 param 传入参数的形式。<br/>
<s:url includeParams = "get">
    <s:param name="id" value="%{'5468'}"/>
</s:url>
<hr>
同时指定 action 属性和 value 属性，且使用 param 传入参数的形式。<br/>
<s:url action="first" value="xxx">
    <s:param name="festName" value="'元旦'"/>
    <s:param name="desc" value="'1 月 1 日元旦节'"/>
</s:url>
```

9．i18n和text标签

i18n 和 text 标签用于为国际化提供支持。i18n 标签用于将一个资源包放入值栈，text 标签用于从资源包中获取消息。代码如下所示。

```
<s:i18n name="ApplicationResources">
    <s:text name="title"/>
</s:i18n>
```

i18n 标签将基名为 ApplicationResources 的资源包放入值栈中，text 标签从资源包中获取键为 title 的文本消息。

i18n 标签和 text 标签的属性表 8-13 和表 8-14 所示。

表 8-13　i18n标签的属性

属性名称	是否必需	默 认 值	类　型	说　明
name	是	无	String	指定要使用的资源包的基名

表 8-14　text标签的属性

属性名称	是否必需	默 认 值	类　型	说　明
id	否	无	String	如果指定了该属性，那么文本内容将不会输出，而是被保存到 OgnlContext 中，在 text 标签结束后，可以通过该属性的值来引用
name	是	无	String	指定要使用的资源包的基名

10. date标签

date 标签用于格式化输出一个日期。除了可以直接格式化输出一个日期外，date 标签还可以计算指定日期和当前时刻之间的时差。

date 标签的属性如表 8-15 所示。

表 8-15　date标签的属性

属性名称	是否必需	默 认 值	类　型	说　明
id	否	无	String	指定引用该元素的 id 值
format	否	无	Boolean	指定日期的格式化样式
name	是	无	String	要格式化的日期值，必须指定为 java.util.Date 的实例
nice	否	false	String	指定是否输出当前日期值与给定的日期值之间的时差，如果为 true，则输出时差

通常，nice 属性和 format 属性不同时指定(不指定 nice 属性时，该属性值为 true，表明输出指定日期和当前时刻的时差)，指定 format 属性用于将指定日期按 format 指定的格式来格式化输出。

如果没有使用 nice 属性，也没有指定 format 属性，那么 date 标签将在国际化资源包中查找 struts.date.format 键，使用这个键的值作为日期的格式化样式；如果这个键不存在，默认将会使用 DateFormat.MEDIUM 格式化样式。

11. debug标签

debug 标签用于辅助调试，它可以在页面中生成一个 Debug 超链接，单击这个超链接，可以查看到 ValueStack 和 ActionContext 中所有的对象。

debug 标签只有一个 id 属性，这个属性并没有太大的意义，仅仅是该元素的一个引用 ID。

在前面讲过的 set 标签的 set.jsp 页面中添加一个<s:debug>标签，可以看到 ValueStack 和 ActionContext 中的内容，如图 8-4 所示。

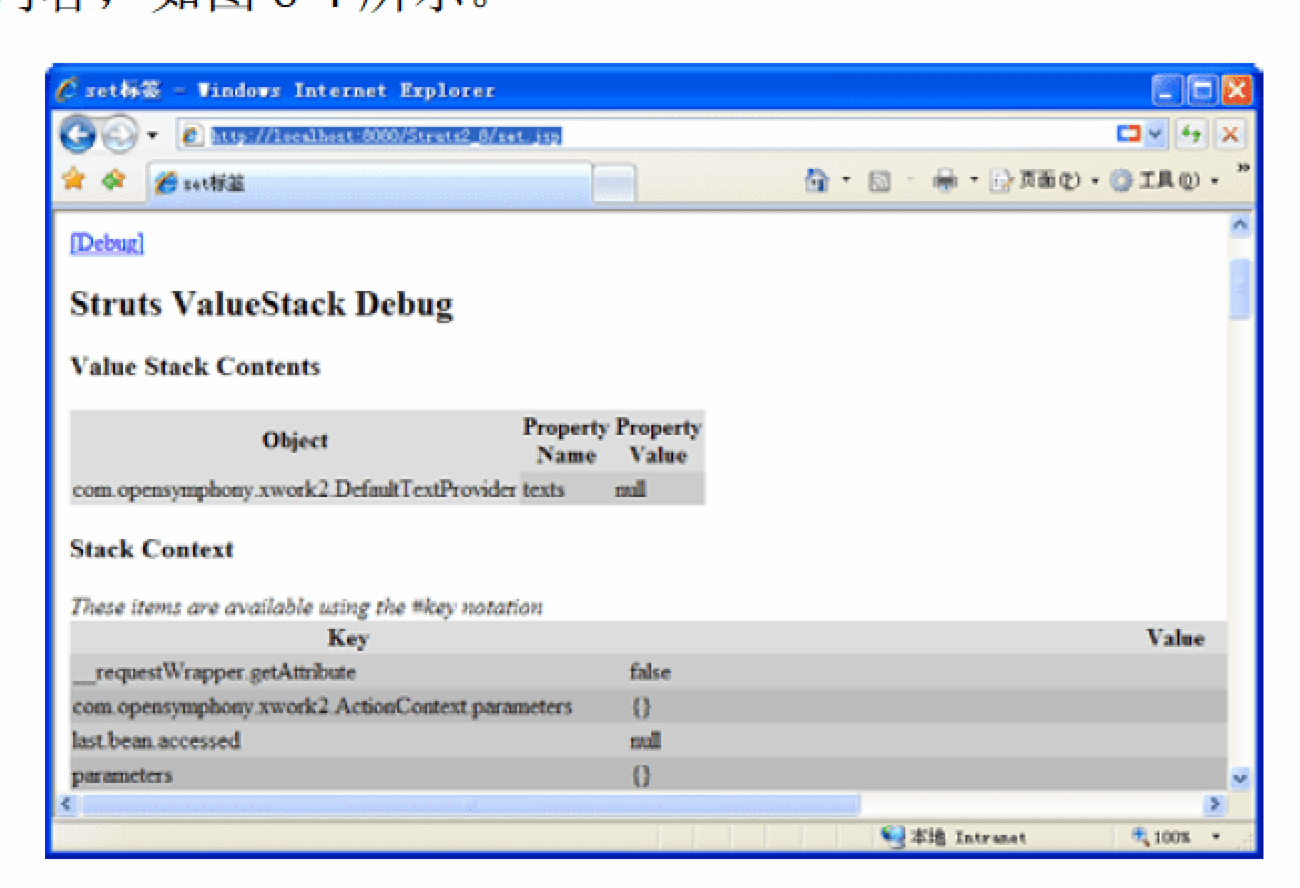


图 8-4　ValueStack和ActionContext中的内容

8.2.2　实例描述

在这个信息化时代，信息就是金钱。如何获取更多的信息，总结信息和运用信息创造财富，被人们时刻关注着。程序也一样，获取数据、处理数据和输出结果数据给用户，是程序的执行主线。Struts 2 提供的数据标签能够使人们更方便快捷地获取数据，处理数据和输出数据。本节实例为读者展示的是，如何运用数据标签来操作显示学员的信息。

8.2.3　实例应用

【例 8-2】 显示学员信息。

(1) 在项目 src/com/struts2/domain 目录下，新建一个 Student 实体类，声明 name(学院名称)、sex(性别)和 favorite(爱好)属性，并分别给出这三个属性的 get 和 set 方法，代码如下所示。

```
public class Student {
    private String name;    //学员名称
    private String sex; //学员性别
    private String favorite;    //爱好

    //下面是上述 name、sex 和 favorite 属性的 get 和 set 方法，在此省略了
}
```

(2) 新建一个 DataAction，继承自 ActionSupport 类，重写 execute()方法，代码如下所示。

```
public class DataAction extends ActionSupport{

    public String execute(){
        //向 Request 对象中放入 info 参数。
        ServletActionContext.getRequest().setAttribute("info", "1001 班成绩最
优异的学员的信息");
        return "success";
    }

}
```

(3) 打开项目 src 目录下的 struts.xml 文件，向该文件中添加 DataAction 配置，代码如下所示。

```
<package name="datatag" extends="struts-default">
    <action name="data" class="com.struts2.action.DataAction">
        <result>/data.jsp</result>
    </action>
</package>
```

(4) 新建一个 JSP 页面 data.jsp，在该页面中使用<s:bean>标签来实例化一个 Student 对象，并使用<s:action>标签访问 DataAction，并获取 Action 中的参数和 Student 对象的属性值显示在页面中，代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
  <%@ taglib uri="/struts-tags" prefix="s" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>My JSP 'data.jsp' starting page</title>
    </head>
    <body>
        <s:bean name="com.struts2.domain.Student" id="student">
            <s:param name="name" value="'刘晓晓'"/>
            <s:param name="sex" value="'女'" />
            <s:param name="favorite" value="'唱歌'"/>
        </s:bean>
        <s:action name="data" executeResult="false"/>
        <s:property value="#attr.info"/><br/>
        学员名字：<s:property value="#student.name"/><br/>
        学员年龄：<s:property value="#student.sex"/><br/>
        学员爱好：<s:property value="#student.favorite"/><br/>
    </body>
</html>
```

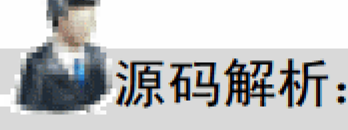
8.2.4 运行结果

代码编写完成后，打开 IE 浏览器，在地址栏中输入“http://localhost:8080/Struts2_8/data.jsp”，在页面中可以看到成绩最优异学员的信息，执行效果如图 8-5 所示。



图 8-5 成绩最优异学员的信息

8.2.5 实例分析

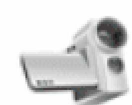


源码解析：

在本实例中，首先封装了一个 Student 实体类，声明 name、sex 和 favorite 属性。然后新建了一个 DataAction，并在 struts.xml 文件中添加了这个 Action 的配置。最后新建了一个 data.jsp 页面，在该页面中使用<s:bean>标签来实例化一个 Student 实例，并为该标签指定了 id="student" 属性，这样我们可以在<s:bean>标签之外访问 Student 实例的各个属性值。此外，还使用<s:action>标签来访问了 DataAction，获取了该 Action 的 Request 对象中的 info 参数，将它显示在页面上。

8.3　主题和模板

主题与模板是 Struts 2 所有 UI 标签的核心，在学习 Struts 2 的 UI 标签之前先来学习一下主题与模板。模板是一个 UI 标签的外在表现形式。如果为所有的 UI 标签都提供了对应的模板，那么这一系列的模板将形成一个主题。

 **视频教学：** 光盘/videos/08/form_property.avi　　长度：13 分钟

8.3.1　基础知识——主题和模板

所有的 UI 标签都是基于模板和主题的。一个模板就是使用 JSP、Velocity 或者 FreeMarker 编写的一个文件，它用于生成 HTML 页面。将一些具有共同观感的模板组织在一起就形成了主题。Struts 2 采用目录名作为主题名，将具有共同观感的模板文件放置在同一个目录下，主题和模板的目录结构如图 8-6 所示。

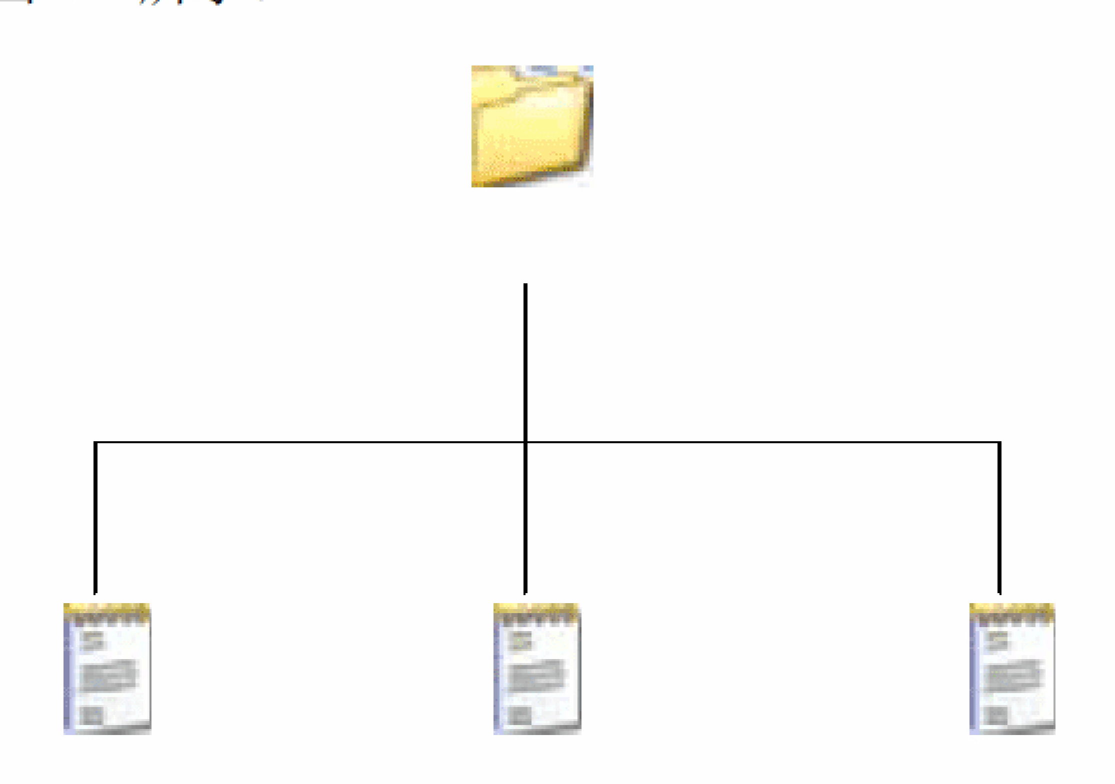


图 8-6　主题与模板的目录结构图

Struts 2 支持三种模板引擎，如下所示。

- ftl(默认)：基于 FreeMarker 的模板引擎。
- vm：基于 Velocity 的模板引擎。
- jsp：基于 JSP 的模板引擎。

可以通过 struts.properties 文件中的 struts.ui.templateSuffix 属性，来配置 Struts 2 使用的默认模板引擎。

1．模板的加载

加载模板主要基于模板路径和主题名。模板路径通过 struts.properties 文件中的 struts.ui.templateDir 属性来配置，该属性的默认值是 template。加载模板时，首先搜索 Web 应用程序根路径下的 template 目录。然后搜索 CLASSPATH 下的 template 目录，如果一个标签使用 xhtml 主题，下面两个位置将被搜索(按顺序)。

- Web 应用程序根路径下 /template/ajax/template.ftl。
- CLASSPATH /template/ajax/template.ftl。

Struts 2 提供了多种方法来配置模板路径，如下所示。

(1) 使用 UI 标签的 templateDir 属性来配置模板路径。

(2) 使用 page 范围的名为 templateDir 的属性来配置模板路径。

(3) 使用 request 范围的名为 templateDir 的属性来配置模板路径。

(4) 使用 session 范围的名为 templateDir 的属性来配置模板路径。

(5) 使用 application 范围的名为 templateDir 的属性来配置模板路径。

(6) 使用 struts.properties 文件中的 struts.ui.templateDir 属性来配置模板路径(默认值为 template)。

上述几种配置模板路径的方法，它们之间存在优先级关系，排列越靠前的方法优先级越高，采用优先级高的方法配置的模板路径，将覆盖采用优先级低的方法配置的模板路径。

2. 选择主题

主题可以按下列的规则进行选择。

(1) 使用 UI 标签的 theme 属性来选择主题。

(2) 使用 UI 标签外围的 form 标签的 theme 属性来选择主题。

(3) 使用 page 范围的名为 theme 的属性来选择主题。

(4) 使用 request 范围的名为 theme 的属性来选择主题。

(5) 使用 session 范围的名为 theme 的属性来选择主题。

(6) 使用 application 范围的名为 theme 的属性来选择主题。

(7) 使用 struts.properties 文件中的 struts.ui.theme 属性来选择主题(默认值是 xhtml)。



修改 form 标签的 theme 属性，可以覆盖整个表单的主题。为用户提供个性化的界面观感时，可以使用用户的 session 来改变主题。修改 struts.properties 文件中的 struts.ui.theme 属性，可以改变整个应用程序的主题。

3. 创建新的主题

有时候，系统提供的主题可能不能完全满足程序开发者的需求。此时需要创建自定义的主题。如果只是想改变某个 UI 标签的呈现方式，只需要覆盖现有标签对应的模板即可，也可以创建新的模板添加到现有的主题。如果需要一套更加丰富可重用的模板，可以创建全新的主题。

Struts 2 提供了三种方式用于创建新的主题，如下所示。

(1) 重新创建一个全新的主题。

(2) 包装一个现有的主题。

(3) 扩展一个现有的主题。

一般不建议重新创建一个全新的主题，建议采用后两种方式。Struts 2 内置提供的最简单的主题是 simple 主题，它给出了 UI 标签最基本的结构，可以在它的基础上包装或扩展一个现有的主题。

Struts 2 提供的 xhtml 主题就是一个利用 simple 主题“包装”而成的新主题。simple 主题提供了基本的控制，xhtml 主题通过添加头部和尾部“装饰”了更多的控制。下面是 xhtml 主题

下的模板使用的一种包装形式。

```
<#include "/${parameters.templateDir}/xhtml/controlheader.ftl" />
<#include "/${parameters.templateDir}/simple/xxx.ftl" />
<#include "/${parameters.templateDir}/xhtml/controlfooter.ftl" />
```

上述模板使用 xhtml 主题下的 controlheader.ftl 和 controlfooter.ftl 模板包装了 simple 主题下的 xxx.ftl 模板。

采用包装的方式创建新主题，需要实现 UI 标签对应的每一个模板，工作量也是相当庞大的。如何才能更加方便快捷地创建一个新主题呢？Struts 2 的主题提供了类似对象继承的能力，即一个主题可以扩展另一个主题，然后重写需要修改的模板，其余的模板将从父模版中加载即可。xhtml 主题不仅采用了包装技术，也采用了扩展技术。Struts 2 内置的 ajax 主题扩展了 xhtml 主题。

要扩展一个主题，需要在主题对应的目录下包含一个 theme.properties 属性文件。在该文件中使用 parent 键指定要扩展的主题名，例如：ajax 主题包含的 theme.properties 文件的内容如下。

```
parent=xhtml
```

8.3.2　基础知识——Struts 2 内置的四种主题

Struts 2 内置的四种主题：simple、xhtml、css_xhtml 和 ajax 主题。本节分别向读者讲解这四种主题的特性。

simple 主题是最底层的结构，提供了简单的 HTML 元素支持，可以用于构建附加的功能或行为。

xhtml 主题是 Struts 2 默认的主题，它对 simple 主题进行了包装和扩展，提供了附加的功能或行为。xhtml 主题还增加了如下的特性。

(1) 针对 HTML 标签(如 textfield 和 select 标签)使用标准的两列表格布局。

(2) 每个 HTML 标签的 Label，即可以出现在 HTML 元素的左边，也可以出现在上边，这取决于 labelposition 属性的设置。

(3) 自动输出校验错误信息。

(4) 输出 JavaScript 的客户端校验。



xhtml 主题输出的表格，有两种布局方式：两列布局方式(label 和表单元素分占两列)和两行布局方式(label 和表单元素分占两行)，取决于表单标签的 labelposition 属性的取值(left 或者 top)。

css_xhtml 主题与 xhtml 主题相似，它也包装了 simple 主题，扩展了 xhtml 主题；但 css_xhtml 主题不是采用表格对表单元素进行布局，而是采用 css 和<div>对表单元素进行布局的。

css_xhtml 主题增加了如下特性。

(1) 针对 HTML 中与表单相关的标签使用标准的两列基于 CSS 和<div>的布局。

(2) 对于每个 HTML 标签的 label，依照 CSS 样式表的设置来决定位置。

(3) 自动输出验证错误。

(4) 输出 JavaScript 的客户端校验。

ajax 主题也对 xhtml 主题进行了扩展，并增加了自己特有的特性。ajax 主题的 Ajax 支持是以 Dojo 和 DWR 为基础的。ajax 主题在 xhtml 主题基础上增加了如下特性。

(1) Ajax 方式的客户端验证。

(2) 支持远程表单的异步提交(最好和 submit 标签一起使用)。

(3) 提供高级的 div 标签，允许动态重新加载部分 HTML 的功能。

(4) 提供高级的 a 标签，允许动态加载并执行远端的 JavaScript 代码。

(5) 支持 Ajax 的 tabbedPanel 实现。

(6) 提供“富客户端”模型的 pub-sub 事件模型。

(7) 交互的 autocomplete 标签。

8.4 个人信息表单

Struts 2 的表单标签可以分为两种：form 标签本身和单个表单元素的标签。form 标签本身的行为不同于自身内部的元素标签。Struts 2 的表单元素标签包含了非常多的属性，但有很多属性完全是公共的属性。本节将在基础知识中，首先讲解表单标签的公共通用属性，然后再分别讲解各个元素标签各自特有的属性。

视频教学： 光盘/videos/08/form_property.avi 长度：13 分钟
光盘/videos/08/form.avi 长度：6 分钟
光盘/videos/08/select.avi 长度：6 分钟
光盘/videos/08/optgroup.avi 长度：7 分钟
光盘/videos/08/radio.avi 长度：6 分钟
光盘/videos/08/combobox.avi 长度：8 分钟
光盘/videos/08/checkboxlist.avi 长度：8 分钟
光盘/videos/08/doubleselct.avi 长度：9 分钟
光盘/videos/08/optiontransferselect.avi 长度：9 分钟
光盘/videos/08/updownselect.avi 长度：7 分钟
光盘/videos/08/submit.avi 长度：10 分钟

8.4.1 基础知识——表单标签

Struts 2 的所有表单标签处理类都继承自 UIBean 类，UIBean 包含了一些通用属性，这些通用属性分为四种。

- 模板相关的属性。
- JavaScript 相关的属性。
- 工具提示相关的属性。
- 通用属性。

除了这些属性外，所有表单元素标签都存在一个特殊的属性：form(${parameters.form})，

这个属性引用表单元素所在的表单，通过 form 属性可以实现表单元素与表单之间的交互。例如：使用${parameters.form.id}获取 form 标签的 id 属性。

(1) 模板相关的属性。

模板相关的通用属性如表 8-16 所示。

表 8-16　模板相关的通用属性

属性名称	数据类型	说　明
templateDir	String	指定表单所用的模板文件目录
theme	String	指定表单所用的主题
template	String	指定表单所用的模板

(2) JavaScript 相关的属性。

JavaScript 相关的通用属性如表 8-17 所示。

表 8-17　JavaScript相关的通用属性

属性名称	主　题	数据类型	说　明
onclick	simple	String	指定鼠标在该标签生成的表单元素上单击时触发的 JavaScript 函数
ondbclick	simple	String	指定鼠标在标签生成的表单元素上双击时触发的 JavaScript 函数
onmouseup	simple	String	指定鼠标在该标签生成的表单元素上松开时触发的 JavaScript 函数
onmousedown	simple	String	指定鼠标在该标签生成的表单元素上按下时触发的 JavaScript 函数
onmouseout	simple	String	指定鼠标移出该标签生成的表单元素时触发的 JavaScript 函数
onmouseover	simple	String	指定鼠标在该标签生成的表单元素上悬停时触发的 JavaScript 函数
onfocus	simple	String	指定该标签生成的表单元素得到焦点时触发的函数
onblur	simple	String	指定该标签生成的表单元素失去焦点时触发的函数
onkeypress	simple	String	指定单击键盘上某个键时触发的函数
onkeyup	simple	String	指定松开键盘上某个键时触发的函数
onkeydown	simple	String	指定按下键盘上某个键时触发的函数
onselect	simple	String	对下拉列表项等可以选择表单元素，指定选中该元素时触发的 JavaScript 函数
onchange	simple	String	对于文本框等可以接受输入的表单元素，指定当值改变时触发的 JavaScript 函数



由于 HTML 元素本身的限制，并不是每个 HTML 元素都可以触发以上的所有函数，因此，上面的属性并不是对 Struts 2 的每个标签都有效。

(3) 工具提示相关的属性。

Struts 2 为表单元素提供了提示功能，当鼠标在表单元素上悬停时，浏览器显示浮动的提示信息。工具提示相关的通用属性如表 8-18 所示。

表 8-18 工具提示相关的通用属性

属性名称	数据类型	说　明
tooltip	String	设置此组件的 Tooltip
tooltipConfig	String	配置工具提示的各种属性

(4) 通用属性。

通用属性如表 8-19 所示。

表 8-19 通用属性

属性名称	主　题	数据类型	说　明
cssClass	simple	String	设置表单元素的 class 属性
cssStyle	simple	String	设置表单元素的 style 属性，使用内联的 CSS 样式
title	simple	String	设置表单元素的 title 属性
disabled	simple	String	设置表单元素的 disabled 属性
label	xhtml	String	设置表单元素的 label 属性
labelPosition	xhtml	String	设置表单元素 label 所在位置，可接受的值为 top(上面)和 left(左边)，默认是在左边
requiredposition	xhtml	String	定义必填标记(默认以*作为必填标记)位于 label 元素的位置，可接受的值为 left(左面)和 right(右边)，默认是在右边
name	simple	String	定义表单元素的 name 属性，该属性值用于与 Action 的属性形成对应
required	xhtml	Boolean	定义是否在表单元素的 label 上增加必填标记(默认以*作为必填标记)，设置为 true 时增加必填标记，否则不增加
tabindex	simple	String	设置表单元素的 tabindex 属性
value	simple	String	设置表单元素的 value 属性
key	simple	String	指定表单元素对应的 action 的属性名。该属性将自动生成 name，label 和 value 属性的值



虽然 UI 标签都支持上述属性，但对于某些标签来说，上述的某些属性是没有意义的或者是不需要的，例如 form 标签支持 tabindex 属性，但是没有任何一个主题呈现这个属性。

1. form标签

form 标签用于输出一个 HTML 输入表单，此外，xhtml 主题的 form 标签还输出表单元素外围的表格。

除了公共属性外，form 标签的常用属性如表 8-20 所示。

表 8-20　form标签的常用属性

属性名称	是否必需	默 认 值	类　型	说　明
action	否	当前的 Action	String	指定提交的 Action 的名字，不要添加.action 后缀
namespace	否	当前的名称空间	String	指定提交的 Action 所属的名称空间
method	否	post	String	HTML 表单的 method 属性，取值为 get 或者 post
enctype	否	无	String	上传文件时，设为 multipart/form-data
focusElement	否	无	String	指定某个表单元素的 id，当页面被加载时，该元素将具有焦点
validate	否	false	Boolean	该属性只有在使用 xhtml 或 ajax 主题时才有效，用于指定是否执行客户端验证

form 标签的用法具体如下所示。

```
<s:form action="login" method="post"/>
```

2. textfield标签

textfield 标签用于输出一个 HTML 单行文本输入控件，相当于 HTML 代码：<input type="text" .../>。

除了公共属性外，textfield 标签的属性如表 8-21 所示。

表 8-21　textfield标签的属性

属性名称	是否必需	默 认 值	类　型	说　明
maxlength	否	无	Integer	文本输入控件可以输入字符的最大长度
readonly	否	false	Boolean	如果该属性值设为 true，用户将不能在文本控件中输入文本
size	否	无	Integer	指定文本输入控件的可视尺寸

使用 form 标签和 textfield 标签实现一个登录表单，代码如下所示。

```
<s:form action="login" method="post">
    <s:textfield name="username" label="用户名"/>
    <s:textfield name="password" label="密码"/>
</s:form>
```

3. password标签

password 标签用于输出一个 HTML 口令输入控件，相当于 HTML 代码：<input type="password" .../>。

除了公共属性外，password 标签的属性如表 8-22 所示。

表 8-22　password标签的属性

属性名称	是否必需	默 认 值	类　型	说　明
maxlength	否	无	Integer	口令输入控件可以输入字符的最大长度
readonly	否	false	Boolean	当该属性的值为 true 时，用户不能在口令控件中输入密码
size	否	无	Integer	指定口令输入控件的可视尺寸
showPassword	否	false	Boolean	是否显示密码。当设为 true 时，密码被显示。一般不要设为 true

password 标签的用法如下。

```
<s:password name="password" label="密码" />
```

4．textarea标签

textarea 标签用于输出一个 HTML 多行文本输入控件，相当于 HTML 代码：<textarea .../>。除了公共属性外，textarea 标签的属性如表 8-23 所示。

表 8-23　textarea标签的属性

属性名称	是否必需	默 认 值	类　型	说　明
cols	否	无	Integer	指定多行文本输入控件的可输入文本的列数
rows	否	无	Integer	指定多行文本输入控件可输入文本的行数
readonly	否	false	Boolean	当该属性的值为 true 时，用户不能在文本输入控件中输入文本
wrap	否	false	Boolean	指定多行文本输入控件中的内容是否应该换行

textarea 标签的用法如下所示。

```
<s:textarea name="dep_desc" cols="50" rows="5" label="部门简介"/>
```

5．select标签

select 标签用于输出一个 HTML 列表框，相当于 HTML 代码。

```
<select …><option …>…</option></select>
```

除了公共属性外，select 标签的属性如表 8-24 所示。

表 8-24　select标签的属性

属性名称	是否必需	默 认 值	类　型	说　明
list	是	无	Collection 、 Map 、Enumeration、Iterator 或者 Array	指定将要迭代的集合，使用该集合中的元素来设置各个选项。如果 list 属性的值是一个 Map，则 Map 的 key 会成为选项的 value，Map 的 value 会成为选项的内容

续表

属性名称	是否必需	默 认 值	类　型	说　明
listKey	否	无	String	指定集合元素中对象的某个属性作为选项的 value
listValue	否	无	String	指定集合元素中对象的某个属性作为选项的内容
headerKey	否	无	String	指定当用户选择了 header 选项时提交的值，如果使用该属性，不能为该属性赋空值
headerValue	否	无	String	指定显示在页面中 header 选项的内容
emptyOption	否	false	Boolean	指定是否在 header 选项后面添加一个空选项
multiple	否	false	Boolean	设置列表框是否允许多选
size	否	无	Integer	设置下拉列表框可显示的选项个数

select 标签的用法如下所示。

```
<s:form>
    <s:select label="性别" name="education" list="{'男','女'}"/>
</s:form>
```

上述代码通过 list 属性直接使用 OGNL 表达式创建了一个列表，它产生的效果与下面代码产生的一样。

```
<select name="sex" id="sex">
    <option value="男">男</option>
    <option value="女">女</option>
</select>
```

select 标签的 list 属性也可以直接使用 OGNL 表达式创建一个 Map。Map 中的 key 作为列表框选项的值， Map 中的 value 是作为列表框选项的内容。代码如下所示。

```
<s:form>
    <s:select label="性别" name="sex" list="#{1:'男',2:'女'}"/>
</s:form>
```

select 标签在客户端浏览器中的输出，代码如下所示。

```
<select name="sex" id="sex">
    <option value="1">男</option>
    <option value="2">女</option>
</select>
```

使用 select 标签的 headerKey 和 headerValue 属性可以设置 header 选项，代码如下所示。

```
<s:form>
    <s:select label="性别" name="sex" list="{'男','女'}"
    headerKey="-1" headerValue="请选择您的性别"/>
</s:form>
```

设置 header 选项主要目的就是用来提示用户操作，因此将 header 选项的值使用 headerKey 属性设为无意义的值，如-1。

6. optgroup标签

optgroup 标签作为 select 标签的子标签，用于创建选项组。可以在 select 标签的标签体中使用一个或者多个 optgroup 标签，对选项进行逻辑分组。但 optgroup 标签自身不能嵌套。

除了公共属性外，optgroup 标签的属性如表 8-25 所示。

表 8-25 optgroup标签的属性

属性名称	是否必需	默 认 值	类 型	说 明
list	是	无	Collection、Map、Enumeration、Iterator 或者 Array	指定将要迭代的集合，使用集合中的元素来设置各个选项。如果 list 属性的值是一个 Map，则 Map 的 key 会成为选项的 value，Map 的 value 会成为选项的内容
listKey	否	无	String	指定集合元素中对象的某个属性作为选项的 value
listValue	否	无	String	指定集合元素中对象的某个属性作为选项的内容

使用 select 标签创建一个下拉列表框，并使 optgroup 标签为 select 标签中的选项进行分组。代码如下所示。

```
<s:form>
    <s:select label="职位" name="position" list="#{1:'经理',2:'主管',3:'代表
'}">
        <s:optgroup label="项目组长" list="#{4:'Java 项目组长',5:'.net 项目组长
'}"/>
        <s:optgroup label="普通员工" list="#{6:'Java 程序员',7:'.net 程序员'}"/>
    </s:select>
</s:form>
```

7. radio标签

radio 标签用于输出一组 HTML 单选按钮，相当于一组 HTML 代码：<input type="radio" .../>

除了公共属性外，radio 标签的属性如表 8-26 所示。

表 8-26 radio标签的属性

属性名称	是否必需	默 认 值	类 型	说 明
list	是	无	Collection、Map、Enumeration、Iterator 或者 Array	指定将要迭代的集合，使用集合中的元素来设置各个选项。如果 list 属性的值是一个 Map，则 Map 的 key 会成为选项的 value，Map 的 value 会成为选项的内容
listKey	否	无	String	指定集合元素中对象的某个属性作为选项的 value
listValue	否	无	String	指定集合元素中对象的某个属性作为选项的内容

radio 标签的使用方式与 select 标签相似，它的具体用法如下所示。

```
<s:form>
    <s:radio name="ismarry" value="1" list="#{1:'已婚',0:'未婚'}" label="是否婚配"/>
</s:form>
```

上述代码中将 radio 标签的 value 属性的值设置为 1，该值将与 list 属性配置的 Map 中的元素进行比较，值为 1 的选项被默认选中。

8．checkbox标签

checkbox 标签用于输出一个 HTML 复选框，相当于 HTML 代码。

```
<input type="checkbox" …/>
```

除了公共属性外，checkbox 标签还有一个 fieldValue 属性，该属性是使用时必需指定的，默认值为 true，指定在复选框选中时，实际提交的值。

如果需要使用 checkbox 标签创建一个 value 属性为 true 或者 false 的复选框。可以通过 checkbox 标签的 fieldValue 属性来指定创建 HTML 复选框 value 属性的值。代码如下所示。

```
<s:checkbox name="protocol" label="是否同意上述条款" fieldValue="true"/>
```



将 fieldValue 属性设置为 false，可能会导致一些问题，所以一般情况下不要将 fieldValue 属性设置为 false。

9．checkboxlist标签

checkboxlist 标签可以创建一系列复选框，属性设置与 select 和 radio 标签相似。除了公共属性外，checkboxlist 标签的属性如表 8-27 所示。

表 8-27　checkboxlist标签的属性

属性名称	是否必需	默 认 值	类　型	说　明
List	是	无	Collection、Map、Enumeration、Iterator 或者 Array	指定将要迭代的集合，集合中的元素用来设置各个选项。如果 list 属性的值是一个 Map，则 Map 的 key 会成为选项的 value，Map 的 value 会成为选项的内容
listKey	否	无	String	指定集合元素中对象的某个属性作为选项的 value
listValue	否	无	String	指定集合中对象的某个属性作为选项的内容

使用 checkboxlist 标签，创建一个系列复选框，用来选取学习科目，用法如下所示。

```
<s:form>
    <s:checkboxlist name="interest" list="{'语文','数学','英语','物理','化学','地理'}" label="学习科目" />
</s:form>
```

10. combobox标签

combobox 标签用于生成一个单行文本框和下拉列表框的组合。两个表单元素只对应一个请求参数，只有单行文本框里的值才包含请求参数，下拉列表框只是用于辅助输入，并没有 name 属性，不会产生请求参数。可以使用列表框将文本放置到文本控件中，也可以直接在文本控件中输入文本。

除了公共属性外，combobox 标签的属性如表 8-28 所示。

表 8-28　combobox标签的属性

属性名称	是否必需	默 认 值	类　型	说　明
List	是	无	Collection、Map、Enumeration、Iterator 或者 Array	要迭代的集合，使用集合中的元素来设置列表框中的各个选项
Maxlength	否	无	Integer	指定组合框的文本控件部分可以输入字符的最大长度
Readonly	否	false	Boolean	当该属性的值为 true 时，用户不能在文本控件中输入文本
Size	否	无	Integer	指定组合框的文本输入控件部分的可视尺寸

使用 combobox 标签创建一个选择娱乐项目的组合选择框，代码如下所示。

```
<s:form>
    <s:combobox
        label="请选择娱乐项目"
        name="play"
        list="{'听音乐','打棒球','跳舞','游泳'}"
        headerKey="-1"
        headerValue="---请选择---"
        emptyOption="true"/>
</s:form>
```

用户可以通过下拉列表框选择娱乐项目，也可以在文本控件中直接输入娱乐项目。

11. doubleselect标签

doubleselect 标签输出关联的两个 HTML 列表框，第二个列表框显示的内容随第一个列表框选中的选项而变化。除了公共属性外，doubleselect 标签的属性如表 8-29 所示。

表 8-29　doubleselect标签

属性名称	是否必需	默 认 值	类　型	说　明
List	是	无	Collection、Map、Enumeration、Iterator 或者 Array	指定将要迭代的集合，集合中的元素用来设置各个选项。如果 list 属性的值是一个 Map，则 Map 的 key 会成为选项的 value，Map 的 value 会成为选项的内容

续表

属性名称	是否必需	默 认 值	类　型	说　明
listKey	否	无	String	指定集合元素中对象的某个属性作为选项的 value
listValue	否	无	String	指定集合中对象的某个属性作为选项的内容
headerKey	否	无	String	指定用户在第一个列表框中选择 header 选项提交的值。使用该属性不能为它赋空值
headerValue	否	无	String	指定第一个列表框的 header 选项的内容
emptyOption	否	false	Boolean	是否在第一个列表框的 header 选项后面添加一个空选项
Multiple	否	false	Boolean	该属性对两个列表框都适用，如果设置为 true，则两个列表框都为多选列表框
Size	否	无	Integer	设置下拉列表框可显示的选项个数，该属性只对第一个列表框起作用
doubleList	是	无	Collection、Map、Enumeration、Iterator 或者 Array	该属性对 list 属性中的每一个元素求值，返回一个迭代的集合
doubleListKey	否	无	String	指定集合元素中对象的某个属性作为选项的 value。该属性只对第二个列表框起作用
doubleListValue	否	无	String	指定集合元素中对象的某个属性作为选项的内容。该属性只对第二个列表框起作用
doubleSize	否	无	Integer	设置下拉列表框可显示的选项个数，该属性只对第二个列表框起作用
doubleName	是	无	String	指定第二个列表框的 name 映射，该属性的值与 Action 的属性对应
doubleValue	否	无	Object	第二个列表框的初始选中项

使用 doubleselect 标签，创建一个选择省份与城市的下拉列表框，代码如下所示。

```
<s:doubleselect label="请选择所在省市"
    name="province"
    list="{'河南省','浙江省'}"
    doubleName="city"
    doubleList="top=='河南省'?{'郑州市','巩义市'}:{'杭州市','温州市'}">
</s:doubleselect>
```

12. optiontransferselect标签

optiontransferselect 标签用于创建一个选项转移列表组件，该标签会生成两个<select .../>标签，并且会生成系列的按钮，这些系列的按钮可以控制选项在两个下拉列表之间移动、升降。当提交表单时，将提交两个列表框中选中的选项。

除了公共属性外，optiontransferselect 标签的属性如表 8-30 所示。

表 8-30 optiontransferselect标签的属性

属性名称	是否必需	默认值	类型	说明
List	是	无	Collection、Map、Enumeration、Iterator 或者 Array	指定要迭代的集合，使用集合中的元素来设置各个选项。如果 list 属性的值是一个 Map，则 Map 的 key 会成为选项的 value，Map 的 value 会成为选项的内容。该属性只对第一个列表框起作用
listKey	否	无	String	指定使用集合中对象的某个属性作为选项的 value。该属性只对第一个列表框起作用
listValue	否	无	String	指定使用集合中对象的某个属性作为选项的内容。该属性只对第一个列表框起作用
headerKey	否	无	String	设置当用户在第一个列表框中选择了 header 选项时提交的值。如果使用该属性，不能为该属性赋空值
headerValue	否	无	String	设置第一个列表框的 header 选项的内容
emptyOption	否	false	Boolean	设置是否在第一个列表框的 header 选项后面添加一个空的选项
multiple	否	true	Boolean	设置是否第一个列表框为多选列表框。设置为 true 表示为多选列表框
size	否	无	Integer	设置下拉列表框可显示的选项个数，该属性只对第一个列表框起作用
doubleId	否	无	String	指定第二个列表框的 id
doubleList	是	无	Collection、Map、Enumeration、Iterator 或者 Array	指定要迭代的集合，使用集合中的元素来设置各个选项。如果 doubleList 属性的值是一个 Map，则 Map 的 key 会成为选项的 value，Map 的 value 会成为选项的内容。该属性只对第二个列表框起作用
doubleListKey	否	无	String	指定使用集合中对象的某个属性作为选项的 value。该属性只对第二个列表框起作用
doubleListValue	否	无	String	指定使用集合中对象的某个属性作为选项的内容。该属性只对第二个列表框起作用

续表

属性名称	是否必需	默 认 值	类　型	说　明
doubleHeaderKey	否	无	String	指定当用户选中第二个列表框中 header 选项时提交的值。如果使用该属性，不能赋为空值
doubleHeaderValue	否	无	String	指定第二个列表框的 header 选项的内容
doubleEmptyOption	否	无	String	是否在第二个列表框的 header 选项后面添加一个空选项
doubleMultiple	否	true	Boolean	该属性如果设置为 true，则第二个列表框为多选列表框
doubleSize	否	无	Integer	设置下拉列表框可显示的选项个数，该属性只对第二个列表框起作用
doubleName	是	无	String	指定第二个列表框的 name 映射，该属性的值与 Action 的属性对应
doubleValue	否	无	Object	第二个列表框的初始选中项
leftTitle	否	无	String	设置左边列表框的标题
rightTitle	否	无	String	设置右边列表框的标题
addToLeftLabel	否	<-	String	设置向左移动的按钮上的文本
addToRightLabel	否	->	String	设置向右移动的按钮上的文本
addAllToLeftLabel	否	<<--	String	设置全部移动到左边的按钮上的文本
addAllToRightLabel	否	-->>	String	设置全部移动到右边的按钮上的文本
selectAllLabel	否	<*>	String	设置全部选择按钮上的文本
leftUpLabel	否	ˆ	String	设置左边列表框的向上移动按钮上的文本
leftDownLabel	否	ˇ	String	设置左边列表框的向下移动按钮上的文本
rightUpLabel	否	ˆ	String	设置右边列表框的向上移动按钮上的文本
rightDownLabel	否	ˇ	String	设置右边列表框的向下移动按钮上的文本
allowAddToLeft	否	true	Boolean	设置是否使用移动到左边的按钮
allowAddToRight	否	true	Boolean	设置是否使用移动到右边的按钮
allowAddAllToLeft	否	true	Boolean	设置是否使用全部移动到左边的按钮
allowAddAllToRight	否	true	Boolean	设置是否使用全部移动到右边的按钮
allowSelectAll	否	true	Boolean	设置是否使用全部选择按钮
allowUpDownOnLeft	否	true	Boolean	设置是否使用左边列表框的上移和下移按钮
allowUpDownOnRight	否	true	Boolean	设置是否使用右边列表框的上移和下移按钮



常常无需指定 id 和 doubleId 属性，因为这两个属性将由 optiontransferselect 标签自动生成。该标签所生成的 id 和 doubleId 分别为 <form_id>_<optiontransferselect_name>和<form_id>_<optiontransferselect_doubleName>。

使用 optiontransferselect 标签分别指定两个简单的集合，来生成两个下拉列表框的列表项，可以选择喜欢的运动。代码如下所示。

```
<s:head/>
<s:form>
    <s:optiontransferselect
        label="最喜欢的运动"
        name="tsports"
        leftTitle="球类运动"
        rightTitle="娱乐运动"
        list="{'打网球','打棒球','踢足球'}"
        headerKey="-1"
        headerValue="---请选择---"
        doubleName="jsports"
        doubleList="{'健身体操','跳绳','踢毽子','游泳'}"
        doubleHeaderKey="-1"
        doubleHeaderValue="---请选择---"
        doubleEmptyOption="true"
        addToLeftLabel="向左移动"
        addToRightLabel="向右移动"
        addAllToLeftLabel="全部左移"
        addAllToRightLabel="全部右移"
        selectAllLabel="全部选择"
        leftUpLabel="向上移动"
        leftDownLabel="向下移动"
        rightUpLabel="向上移动"
        rightDownLabel="向下移动"
    />
</s:form>
```

上述实例使用了 optiontransferselect 标签的大部分常用属性，比如：用来控制选项移动的按钮、按钮上的文本设置，等等。

13．label标签

xhtml 主题提供的 label 标签输出两个 HTML 的 label 标签，分别位于一行的两列，左列的 label 标签起提示作用，右列的 label 标签用于显示只读的 action 属性数据。而 simple 主题提供的 label 标签只输出一个 HTML label 标签。

label 标签的基本用法如下。

```
<s:label label="用户名" name="username"/>
```

如果请求的 Action 创建了实例，并且 username 属性有值，将在右列的标签上显示 username 属性的值。

14．file标签

file 标签用于输出一个 HTML 文件选择框，相当于 HTML 代码：<input type="file" .../>。

除了公共属性外，file 标签特有的属性为 accept，默认值为 input，这个属性可以指出接受的文件的 MIME 类型。

file 标签的基本用法如下。

```
<s:file name="uploadFile" accept="text/*" />
```

15．head标签

head 标签输出对应主题的 HEAD 部分的内容。如果有些主题需要包含特定的 CSS 和 JavaScript，可以使用 head 标签来输出这些代码。

除了公共属性外，head 标签特有的属性如表 8-31 所示。

表 8-31　head标签的属性

属性名称	是否必需	默 认 值	类　型	说　明
debug	否	无	String	如果使用 ajax 主题时将该属性设置为 true，那么将开启调试模式
Calendarcss	否	无	String	jscalendar 控件使用的 css 主题

head 标签的基本用法如下。

```
<html>
    <head>
        <title>head 标签</title>
    </head>
    <body>
        <s:head/>
    </body>
</html>
```

16．token标签

token 标签用于防止多次提交表单(避免刷新页面时多次提交)，如果需要该标签起作用，则应该在 Struts 2 的配置文件中启用 TokenInterceptor 拦截器，获知 TokenSessionStoreInterceptor 拦截器。

token 标签的实现原理是在表单中添加一个隐藏域，每次加载该页面时，该隐藏域的值都不相同。TokenInterceptor 拦截器拦截所有用户请求，如果两次请求时该 token 对应隐藏域的值相同，则阻止表单提交。

通过上面的介绍可以看出，token 标签无需在页面上生成任何输出，也无需开发者手动控制，因此使用该标签无需指定任何属性。



在默认情况下，token 标签生成的隐藏域的 name 为 struts.token。因此，不要在表单中再定义一个名为 struts.token 的表单域。

token 标签生成一个阻止重复提交的隐藏域，基本用法如下。

```
<s:token/>
```

17．updownselect标签

updownselect 标签的用法类似于 select 标签，区别是该标签生成的列表框可以支持选项的

上下移动。因此使用该标签时，一样可以指定 list、listKey 和 listValue 等属性，这些属性的作用与使用 select 标签时指定的 list、listKey 和 listValue 等属性完全相同。当提交表单时，列表框中选中的选项将被提交。除了公共属性外，updownselect 标签特有的属性如表 8-32 所示。

表 8-32　updownselect标签的属性

属性名称	是否必需	默 认 值	类　型	说　明
list	是	无	Collection、Map、Enumeration 、Iterator 或者 Array	指定要迭代的集合，使用集合中的元素来设置各个选项。如果 list 属性的值是一个 Map，则 Map 的 key 会成为选项的 value，Map 的 value 会成为选项的内容
listKey	否	无	String	指定使用集合中对象的某个属性作为选项的内容。该属性只对第一个列表框起作用
listValue	否	无	String	指定使用集合中对象的某个属性作为选项的 value。该属性只对第一个列表框起作用
headerKey	否	无	String	指定当用户选择了 header 选项时提交的值，如果使用该属性，不能为该属性赋空值
headerValue	否	无	String	设置 header 选项的内容
emptyOption	否	false	Boolean	是否在 header 选项后面添加一个空选项
multiple	否	true	Boolean	如果设置为 true，则创建一个多选列表
size	否	无	Integer	指定下拉框可显示的选项个数
moveUpLabel	否	ˆ	String	设置“上移”按钮上的文本，默认是ˆ符号
moveDownLabel	否	ˇ	String	设置“下移”按钮上的文本，默认是ˇ符号
selectAllLabel	否	*	String	设置“全选”按钮上的文本，默认是*符号
allowMoveUp	否	无	String	是否显示“上移”按钮，默认为 true
allowMoveDown	否	无	Boolean	是否显示“下移”按钮，默认为 true
allowSelectAll	否	无	Boolean	是否显示“全选”按钮，默认为 true

使用 updownselect 标签创建一个可以上下移动选项的列表框，代码如下所示。

```
<s:head/>
<s:form>
    <s:updownselect
        list="{'打网球','打棒球','踢足球'}"
        name="tsports"
        headerKey="-1"
        headerValue="---请选择---"
        emptyOption="true"
        moveUpLabel="上移"
```

```
        moveDownLabel="下移"
        selectAllLabel="全选"
        />
</s:form>
```

18. hidden标签

hidden 标签用于输出一个 HTML 隐藏表单元素，相当于 HTML 代码：<input type="hidden" …/>。xhtml 主题直接从 simple 主题继承了 hidden.ftl 模板。xhtml 主题中 hidden 标签与其他的标签不太一样，hidden 标签不输出表行。另外，hidden 标签除了公共属性外，没有自己特有的属性。

hidden 标签的基本用法如下所示。

```
<s:hidden name="username" value="Jack" />
```

19. submit标签

submit 标签用于输出一个提交按钮。submit 标签和 form 标签一起使用可以提供异步表单提交功能。submit 标签可以输出以下三种类型的提交按钮：input、image 和 button。

除了公共属性外，submit 标签自己特有的属性如表 8-33 所示。

表 8-33　submit标签的属性

属性名称	是否必需	默 认 值	类　型	说　明
type	否	input	String	设置提交按钮的类型，可选值：input、button、image
src	否	无	String	设置 image 类型的提交按钮的图片地址，该属性对 input 和 button 类型的提交按钮无效
action	否	无	String	指定处理请求的 Action
method	否	无	String	指定处理请求的 Action 的方法

创建一个 input 类型的提交按钮，并使用 action 和 method 属性分别指定处理请求的 Action 和处理请求的 Action 的方法。代码如下所示。

```
<s:submit type="input" action="BookManager" method="addBook" label="添加"/>
```

创建 button 类型的提交按钮，与 input 类型的差不多，代码如下所示。

```
<s:submit type="button" action="BookManager" method="addBook" label="添加"/>
```

创建 image 类型的提交按钮，使用 method 属性，代码如下所示。

```
<s:submit type="image" method=" register " src="images/register.gif" />
```

20. reset标签

reset 标签用于输出一个重置按钮，一般与 form 标签结合使用提供表单的重置功能。

除了公共属性外，reset 标签自己特有的属性有 type，它的默认值是 input，可以用来指定重置按钮的类型，可选值为 input 和 button。

reset 标签的基本用法如下。

```
<s:reset value="重置"/>     <!—默认值按钮类型为 input -->
<s:reset type="button" label="重置"/>
```

如果采用默认的 input 重置按钮类型，只能通过 value 属性来设置重置按钮上的文本。

8.4.2　实例描述

页面中表单应用是非常频繁的，比如：提交一些用户注册信息、用户登录信息，等等。但使用一般的 HTML 代码实现看起来又太凌乱，因此借助于 Struts 2 提供的表单标签，可以更灵活地生成各种表单。本节实例将介绍如何使用 Struts 2 表单标签来实现个人信息表单。

8.4.3　实例应用

【例 8-3】 个人信息表单。

(1) 新建一个 PersonFormAction 类，声明 username(用户名)、password(密码)、truename(真实姓名)、sex(性别)、education(学历)、province(省份)、city(城市)和 favorite(爱好)属性，并分别为这些属性创建 get、set 方法，重写继承于 ActionSupport 类的 execute()方法，代码如下所示。

```
public class PersonFormAction extends ActionSupport {
    private String username;    //用户名
    private String password;    //密码
    private String truename;    //真实姓名
    private String sex;     //性别
    private String education;   //学历
    private String province;    //省份
    private String city;        //城市
    private String favorite;        //爱好

    //下面是属性 username、password、truename、sex、education、province、city 和
favorite 的 get
    //和 set 方法，在此省略了。

    public String execute() throws Exception{
        return "success";
    }
}
```

(2) 打开项目目录 src/struts.xml 文件，向该文件中添加 PersonFormAction 配置，代码如下所示。

```
<package name="formtag" extends="struts-default">
    <action name="personform" class="com.struts2.action.PersonFormAction">
        <result>/personinfo.jsp</result>
    </action>
</package>
```

(3) 新建一个 formel.jsp 页面，在该页面中使用<s:form>标签创建一个 form 表单，并使用表单元素标签创建一些表单元素(文本输入框、单选按钮、下拉列表框，等等)，以供用户输入或选择相关信息。代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>表单标签的应用</title>
    </head>
    <body>
        <s:form action="personform">
            <s:textfield name="username" label="用户名" />
            <s:password name="password" label="密码"/>
            <s:textfield name="truename" label="真实姓名" />
            <s:radio name="sex" list="{'男','女'}" label="性别"/>
            <s:select name="education" label="最高学历"
                list="{'高中','大学','硕士','博士'}">
            </s:select>
            <s:doubleselect label="请选择所在省市"
                name="province"
                list="{'河南省','浙江省'}"
                doubleName="city"
                doubleList="top=='河南省'?{'郑州市','巩义市'}:{'杭州市','温州市
'}">
            </s:doubleselect>
            <s:checkboxlist name="favorite"
                list="{'书法','听音乐','跳舞','看书','弹钢琴'}"
                label="个人爱好" >
            </s:checkboxlist>
            <s:submit value="提交"/>
        </s:form>
    </body>
</html>
```

(4) 新建一个 personinfo.jsp 页面，用于显示用户提交的个人信息，代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>个人信息</title>
    </head>
    <body>
        你提交的个人信息如下所示：
        <table>
            <tr>
                <td>用户名：</td>
                <td><s:property value="username"/></td>
```

```
            </tr>
            <tr>
                <td>密码：</td>
                <td><s:property value="password"/></td>
            </tr>
            <tr>
                <td>真实姓名：</td>
                <td><s:property value="truename"/></td>
            </tr>
            <tr>
                <td>性别：</td>
                <td><s:property value="sex"/></td>
            </tr>
            <tr>
                <td>学历：</td>
                <td><s:property value="education"/></td>
            </tr>
            <tr>
                <td>城市：</td>
            <td><s:property value="city"/></td>
            </tr>
            <tr>
                <td>爱好：</td>
                <td><s:property value="favorite"/></td>
            </tr>
        </table>
    </body>
</html>
```

8.4.4 运行结果

打开 IE 浏览器，在地址栏中输入“http://localhost:8080/Struts2_8/formel.jsp”，用户可以在该页面中输入或选择个人信息，运行结果如图 8-7 所示。

当用户输入个人信息完毕，单击表单的“提交”按钮，将表单提交到 personform.action 上，该 Action 处理数据成功之后，将跳转到 personinfo.jsp 上，显示用户的个人信息，执行效果如图 8-8 所示。



图 8-7 个人信息表单



图 8-8 个人信息

8.4.5　实例分析



源码解析：

在本实例新建的 formel.jsp 页面中，使用了 form 标签和表单元素标签(textfield 标签、password 标签、select 标签，等等)以供用户填写个人信息。

创建了一个 PersonFormAction，当用户提交表单时，将提交给 PersonFormAction，该 Action 接受了用户提交过来的个人信息，并将这些信息赋给自己的属性(username、password，等等)。当用户请求 PersonFormAction 成功之后，将跳转到 personinfo.jsp 页面，在该页面上使用<s:property>标签将用户信息显示出来。

8.5　选择自己喜欢的节日

非表单标签主要用于在页面中生成一些非表单的可视化元素，例如 Tab 页面、输出 HTML 页面的树形结构等。当然，非表单标签也包含在页面显示 Action 里封装的信息。



视频教学： 光盘/videos/08/component.avi　　长度：7 分钟

光盘/videos/08/actionmessage.avi　　长度：4 分钟

8.5.1　基础知识——非表单标签

非表单标签包括：component、a、actionerror、actionmessage，等等。本节将逐一向读者讲解这些标签的功能及使用方法。

1．component标签

component 标签用于使用自定义的组件，这是一个非常灵活的用法。如果开发者经常需要使用某个效果片段，就可以考虑将这个效果片段定义成一个自定义组件，然后在页面中使用 component 标签来使用该自定义组件。

由于使用自定义组件是基于主题与模板管理的，因此在使用 component 标签时，通常需要指定如表 8-34 所示的属性。

表 8-34　component标签的属性

属性名称	说　明
theme	自定义组件所使用的主题，如果不指定该属性，则默认使用 xhtml 主题
templateDir	指定自定义组件的主题目录，如果不指定，则默认使用系统的主题目录，即 template 目录
template	指定自定义组件所使用的模板

如果模板需要传入某些外部对象，可以在 component 标签内部使用 param 子标签来传递这些对象。如果在模板中还需要取得该参数，可以采用${parameters.key}或者${parameters.get('key')}。如果是 JSP 模板，可以通过<s:property value="%{parameters.key}"/>或者<s:property value="%{parameters.get('key')}"/>来获取对象。

在 Struts 2 中，自定义组件可以使用 FreeMarker、JSP 或者 Velocity 来编写。通过文件的后缀可以找到正确的呈现引擎。



如果使用 JSP 页面作为模板，那么 JSP 模板文件必须位于 Web 应用程序本身的目录中，而不能放到 ClASSPATH 下，否则将找不到 JSP 模板。FreeMarker 和 Velocity 模板则没有这个限制。

2. a标签

a 标签用于创建一个 HTML 超链接。它是为更好地与 ajax 主题一起使用而设计的，也可以在 simple、xhtml 和其他主题中使用。

a 标签提供了一个 String 类型的 href 属性，该属性可以指定链接的 URL 地址。例如下面所示。

```
<s:a href="addUser.action">添加用户</s:a>
```

3. actionmessage、actionerror和fielderror标签

actionmessage、actionerror 和 fielderror 标签用法几乎完全一样。它们都是用于输出消息的，区别是 actionmessage 用于输出 Action 实例的一般性消息，actionerror 标签用于输出 Action 实例的错误消息，fielderror 标签用于输出 Action 实例字段的错误消息。

actionmessage 标签输出的是，Action 实例的一个 Collection 类型的属性 actionMessages 保存的消息，actionerror 标签输出的是，Action 实例的一个 Collection 类型的属性 actionErrors 保存的错误消息，fielderror 标签输出的是，Action 实例的一个 Map 类型的属性 fieldErrors 保存的字段错误信息。

下面的例子中首先新建一个 ErrorAction 类，在这个 Action 中添加一些 Action 的一般消息、Action 的错误消息和 Action 的字段错误消息。代码如下所示。

```
public class ErrorAction extends ActionSupport{
    public String execute() throws Exception{
        //添加 Action 的一般性消息
        addActionMessage("One Action Message");
        addActionMessage("Two Action Message");
        //添加 Action 的错误消息
        addActionError("One Error Action");
        addActionError("Two Error Action");
        //添加 Action 字段的错误消息
        addFieldError("one_Field","One Field Error");
        addFieldError("two_Field","Two Field Error");
        return "success";
    }
}
```

接下来编写一个 JSP 页面，使用 actionmessage、actionerror 和 fielderror 标签分别输出 ErrorAction 中添加的消息，代码如下所示。

```
<h3>输出 Action 的一般性消息</h3>
<s:actionmessage/>
<h3>输出 Action 的错误消息</h3>
<s:actionerror/>
<h3>输出 Action 所有字段的错误消息</h3>
<s:fielderror/>
<h3>输出 Action 的 one_Field 字段的错误消息</h3>
<s:fielderror>
    <s:param value="'one_Field'"></s:param>
</s:fielderror>
```

8.5.2　实例描述

在 Struts 2 中，非表单标签中的 component 标签提供了一个非常灵活的用法，即使自定义组件也是如此。开发者可以根据自己的需要编写模板文件，从而使页面显示更简洁美观、并能提高开发者的工作效率。在本节实例中，将向读者介绍如何使用自定义组件。

8.5.3　实例应用

【例 8-4】 选择自己喜欢的节日。

(1) 首先新建一个 Component 项目，然后在该项目的 Component 目录下创建 dayTemplateDir 目录和 dayTheme 目录，最后在该目录下新建一个 FreeMarker 模板文件：ftlDayTemplate.jsp 文件，在该文件中写入如下代码。

```
<div style="color:red;">
    <@s.select list="parameters.list"/>
</div>
```

(2) 在项目 dayTemplateDir/dayTheme 目录下，新建一个 jspDayTemplate.jsp 模板文件，在该文件中写入如下代码。

```
<%@ page contentType="text/html; charset=UTF-8" language="java"%>
<%@taglib prefix="s" uri="/struts-tags" %>
<div style="color:green;">
    <b>JSP 自定义模板<br>
    请选择您喜欢的节日<br></b>
    <s:select list="parameters.list"/>
</div>
```

(3) 在项目 Component 目录下依次创建 template/xhtml 目录，在 xhtml 目录下新建 myTemplate.jsp 模板文件，在文件中写入如下代码。

```
<%@ page contentType="text/html; charset=UTF-8" language="java"%>
<%@taglib prefix="s" uri="/struts-tags" %>
```

```
<div style="color:red;">
    <b>JSP 自定义模板<br>
    请选择您喜欢的节日<br></b>
    <s:select list="parameters.list"/>
</div>
```

(4) 在项目 Component 目录下新建一个 component.jsp 页面，在该页面中使用 component 标签输出 ftlDayTemplate.ftl 模板、jspDayTemplate.jsp 模板和 myTemplate.jsp 模板。代码如下所示。

```
<%@ page contentType="text/html; charset=UTF-8" language="java"%>
<%@taglib prefix="s" uri="/struts-tags"%>
<html>
    <head>
        <title>使用 s:component 标签</title>
    </head>
    <body>
        <h3>使用 s:component 标签</h3>
        自定义主题与目录<br/>
        FreeMarker 自定义模板<br/>
        请选择您喜欢的节日<br/>
        <s:component
            theme="dayTheme"
            templateDir="dayTemplateDir"
            template="ftlDayTemplate">
        <s:param name="list"
            value="{'春节','元宵节','端午节','七夕节','中秋节','重阳节'}" />
        </s:component>
        <hr/>
        从 Web 应用根路径下加载模板，使用 JSP 模板。
        <s:component
            theme="dayTheme"
            templateDir="dayTemplateDir"
            template="jspDayTemplate.jsp">
        <s:param name="list"
            value="{'春节','元宵节','端午节','七夕节','中秋节','重阳节'}" />
        </s:component>
        <hr/>
        使用默认主题(xhtml)，默认主题目录(template)<br/>
        从 Web 应用中加载模板，使用 JSP 模板。
        <s:component template="mytemplate.jsp">
            <s:param name="list"
                value="{'春节','元宵节','端午节','七夕节','中秋节','重阳节'}" />
        </s:component>
        <hr/>
        使用自定义主题，自定义主题目录<br/>
        从/WEB-INF/classes 路径下加载模板，使用 ftl 模板。
        <s:component
            theme="myTheme"
```

```
              templateDir="myTemplateDir"
              template="myAnotherTemplate">
              <s:param name="list"
                  value="{'春节','元宵节','端午节','七夕节','中秋节','重阳节'}" />
          </s:component>
      </body>
</html>
```

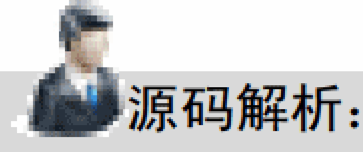
8.5.4　运行结果

打开 IE 浏览器，在地址栏中输入“http://localhost:8080/Component/component.jsp”，执行效果如图 8-9 所示。



图 8-9　使用component标签自定义组件

8.5.5　实例分析



源码解析：

上述实例自定义了一个 FreeMarker 模板文件，ftlDayTemplate.ftl、JSP 模板文件和 jspDayTemplate.jsp，使用默认主题 xhtml 定义了一个 JSP 模板文件 myTemplate.jsp，在 ftlDayTemplate.ftl 文件中指定了文本字体颜色为红色，并接受参数 list 节日列表，使用<@s.select/>来加载 list 列表数据，使用户可以选择自己喜欢的节日。jspDayTemplate.jsp 模板文件中指定了文本字体颜色为绿色，同样，获取参数 list 节日列表，使用<s:select>标签来加载 list 列表数据，myTemplate.jsp 文件和 jspDayTemplate.jsp 差不多，不过指定文件字体颜色为红色。

有了前面的三个模板文件，在 component.jsp 文件中就可以使用 component 标签来简单地引用这些模板，在页面上生成节日下拉选择框，显示不同的文本字体颜色，这就是模板的作用。

8.6 常见问题解答

8.6.1 Struts 2 一遇到标签就出错



Struts 2 一遇到标签就出错，怎么回事？
网络课堂：http://bbs.itzcn.com/thread-10925-1-1.html

问：编写 JSP 程序，但是一遇到 Struts 2 标签就出现如下错误。

```
HTTP Status 500 -
type Exception report
message
description The server encountered an internal error () that prevented it from
fulfilling this request.
exception
org.apache.jasper.JasperException: The Struts dispatcher cannot be found.  This
is usually caused by using Struts tags without the associated filter. Struts
tags are only usable when the request has passed through its servlet filter,
which initializes the Struts dispatcher needed for this tag. - [unknown location]
```

【解决办法】：你的情况是由于 Struts 2 的标签错误导致的，所以首先确保在开头有如下代码。

```
<%@ taglib prefix="s" uri="/struts-tags" %>
```

其次保证必要的 jar 包的导入，最后保证不要重复导入 jar 包，也就是先删除 lib 下的所有 jar 包，再重新导入。

8.6.2 Struts标签库导入错误



Struts 标签库导入错误？
网络课堂：http://bbs.itzcn.com/thread-10927-1-1.html

问：做网站项目练习，使用的SSH，在导入Struts 2 标签库struts-tags.tld时，出现红叉，不能正常使用，应该怎样导入呢？我看见http://diaolanshan.javaeye.com/blog/243956说了一些方法，但是我的是myeclipse，不是eclipse，其实windows-preference中没有amateras啊，应该怎么导？

【解决办法】：Struts 2 的标签库是不用导的，已经存在于 Struts 2 的包里面了。在struts2-core-2.xxxx.jar 下的 META-INF 下面，只要配置 web.xml 中 Struts 2 本来的配置就可以了！使用的时候就包含如下代码。

```
<%@ taglib uri="/struts-tags" prefix="s" %>
```

8.6.3 iterator标签如何循环遍历某一实体下的set集合数据



iterator 标签如何循环遍历某一实体下的 set 集合数据？
网络课堂：http://bbs.itzcn.com/thread-10931-1-1.html

问：本人初学 Struts 2 的标签不是很会使用，希望各位高手能给个使用<s:iterator>标签来循环遍历某一实体下的 set 集合数据。

【解决办法】：我看你初学，就给你写个吧。

Pojo 实体类 User。

```
public class User
{
    int id;
    string name;
    int age;
    string address;

    //getter and setter methods;
}
```

Dao 层代码如下。

```
Set<User> users = userDao.getAllUser();//返回所有的 user 对象，并封装成 Set 集合
```

Action 类代码如下。

```
Set <User> users;
UserDao userDao;
//getter and setter
public String getAllUser()
{
    users = userDao.getAllUser();
    return "success";
}
```

jsp 页面代码如下。

```
<table>
    <tr>
        th>用户 ID</th>
        th>用户名</th>
        <th>用户年龄</th>
        <th>用户地址</th>
    </tr>
    <s:iterator value="users">
        <tr>
            <th>${id}</th>
            <th>${name}</th>
            <th>${age}</th>
```

```
            <th>${address}</th>
        </tr>
    </s:iterator>
</table>
```

8.6.4 使用Struts 2 的bean标签出错



使用 Struts 2 的 bean 标签出错了，求高手帮忙？
网络课堂：http://bbs.itzcn.com/thread-10935-1-1.html

问：在 Action 中得到了某个对象，该对象有两个属性，其中 first=20, last=25,请问使用什么方法可以在页面上输出如下数组：20，21，22，23，24，25。

该对象名为 pager，我的写法如下，请问为什么会报错呢？哪里不对？

```
<s:bean name="org.apache.struts2.util.Counter" id="counter">
    <s:param name="first" value="#pager.first" />
    <s:param name="last" value="#pager.last" />
    <s:iterator>
        counter:<s:property/>
    </s:iterator>
</s:bean>
```

【解决办法】：应该写如下代码。

```
<s:bean name="org.apache.struts2.util.Counter" id="counter">
    <s:param name="first" value="20" />
    <s:param name="last" value="25" />
    <s:iterator>
        counter:<s:property/>
    </s:iterator>
</s:bean>
```

8.6.5 Struts 2 的验证框架，用的是哪个标签返回错误信息



Struts 2 的验证框架，用的是哪个标签返回错误信息？
网络课堂：http://bbs.itzcn.com/thread-2976-1-1.html

问：本人初学 Struts 2 框架，对其标签的使用不清楚，请问<s:actionerror/>、<s:actionmessage/>和<s:fielderror/>这三个标签分别是干什么用的？

Struts 2 的验证框架，用的是哪个标签返回错误信息？

【解决办法】：<s:actionerror/>标签如果在 Action 的方法中通过 addActionError()方法添加错误信息了，那么此标签可以将 actionerror 中的错误信息显示。actionerror，即 Action 级别的错误。

<s:fielderror/>标签如果通过 addFieldError()方法添加错误，此标签可以将其显示出来，fielderror，即字段级错误。

以上两个标签都不推荐使用。

<s:actionmessage/>才是正解，可以通过它来获取 Action 的方法中返回的消息。

Struts 2 的验证框架用的是<s:fielderror>标签返回错误信息。

8.6.6 <s:iterator>标签循环遍历list无法取出类型为类的属性提示ognl.NoConversionPossible错误



<s:iterator>标签循环遍历 list 无法取出类型为类的属性提示 ognl.NoConversionPossible 错误？

网络课堂：http://bbs.itzcn.com/thread-10994-1-1.html

问：初学 Struts 2，遇到一个标签问题，求助高手！

一个 action 里面定义了一个 List，内容是我从后台取出的一个类的集合。该类中包含了一个自定义类型的属性。在前台用<s:iterator>标签循环遍历这个 list，可以取出其中的类，和类的属性，但是那个类型为类的属性就无法取得，提示 ognl.NoConversionPossible，希望各位高手可以指点一二！

【解决办法】：你需要用两层嵌套，第二层是那个类的循环。这是我写过的一个 jsp，核心代码如下所示。

```
<table border="0" cellpadding="5" cellspacing="1" bgcolor="#9AD6FA"
width="100%">
    <s:iterator value="dubList">
        <tr>
            <td bgcolor="#2B6DA4" colspan="4">
                <span class="style7">
                    <s:property value="name"/>
                </span>
            </td>
        </tr>
        <s:iterator value="list">
            <tr bgcolor="#3383C3">
                <td align="center" class="style8"><s:property
value="name"/></td>
                <td align="center" class="style8">
                    <s:if test="role==''||role==null">
                    </s:if>
                    <s:property value="role"/>
                </td>
                <td align="center" class="style8">
                    <s:if test="mail==''||mail==null">---</s:if>
                    <s:property value="mail"/>
                </td>
                <td align="center" class="style8">
                    <s:if test="status"><s:text
name="info.back.through"/></s:if>
```

```
                    <s:else><s:text name="info.back.unexam"/></s:else>
                </td>
            </tr>
        </s:iterator>
    </s:iterator>
</table>
```

8.6.7 Struts 2 在iterator中嵌套radio时，radio标签该怎么写



Struts 2 在 iterator 中嵌套 radio 时，radio 标签该怎么写

网络课堂：http://bbs.itzcn.com/thread-10995-1-1.html

问：初学 Struts 2，想在 iterator 中嵌套使用 radio 标签，使用 HTML 实现代码如下。

```
<tr>
    <td width="10"><input type="radio" name="dc" value="1"></td>
    <td>00001</td>
    <td>数据中心 01</td>
</tr>
<tr>
    <td width="10"><input type="radio" name="dc" value="2"></td>
    <td>00002</td>
    <td>数据中心 02</td>
</tr>
```

现在想用 Struts 2 标签实现上述效果，现在改成 Struts 2 标签，代码写成如下所示。

```
<s:iterator value="dcList" var="item" >
    <tr>
        <td width="10">
            <s:radio name="item.dcradio" list="#item.dcradio"></s:radio>
        <td><s:property value="idcode"/></td>
        <td><s:property value="name"/></td>
    </tr>
</s:iterator>
```

action 里定义 dcradio 为 boolean 型，编写代码后，显示画面的时候会出现个 false，请高手帮帮忙解决。

【解决办法】：你在循环里面生成的 radio 的 name 名称不能都一样。如果 name 都一样的话，就会出现对所有相同 name 名称的 radio 只能选择一个的情况。所以必须给不同的 radio 指定不同的 name。

```
<s:iterator value="dcList" var="item" indexId="i">
    <tr>
        <td width="10"><s:radio name="item.dcradio${i}"
list="#item.dcradio"></s:radio>
        <td><s:property value="idcode"/></td>
        <td><s:property value="name"/></td>
    </tr>
</s:iterator>
```

8.7　习　　题

一、填空题

(1) Struts 2 标签库中的标签大体可以分为四种，分别是数据标签、__________、表单标签和非表单标签。

(2) sort 标签用于对指定的集合元素进行排序，进行排序时，必须提供自己的排序规则，即使实现自己的 Comparator，自己的 Comparator 需要实现__________接口。

(3) property 标签的作用是输出 value 属性指定的值，如果没有指定 value 属性，则默认输出__________的值。

(4) set 标签用于将某个值放入指定范围内，例如 application 范围、session 范围等。我们可以使它的__________属性来指定所要放的范围。

(5) Struts 2 内置的四种主题分别是 simple、__________、css_xhtml 和 ajax 主题。

二、选择题

(1) 控制标签可以完成输出流程控制，例如分支、循环等操作，也可完成对集合的合并、排序等操作。下面属于控制标签的标签有哪些？__________

A. a 标签　　B. if/elseif/else 标签　　C. html 标签

D. submit 标签　　E. url 标签　　F. html 标签

(2) append 标签可以将多个集合对象拼接起来，组成一个新的集合。下面哪个标签可以为它提供需要合并的多个子集合呢？__________

A. iterator 标签　　B. merge 标签

C. subset 标签　　D. param 标签

(3) 数据标签用于访问 ActionContext 和值栈中的数据。下面哪个数据标签可以将某个值放到 ValueStack 的栈顶？__________

A. push 标签　　B. sort 标签

C. set 标签　　D. property 标签

(4) 下面对<s:bean>标签的作用描述正确的是__________。

A. 引入一个某个类　　B. 创建一个 JavaBean 实例

C. 为 JavaBean 实例的属性赋值　　D. 为其他标签提供参数

(5) 通过指定 Action 的名字和可选的名称空间，action 标签允许你在 JSP 页面中直接调用 Action。该标签的__________属性用来指定要调用的 Action。

A. action　　B. value

C. name　　D. id

(6) 下列__________标签用于将一个 JSP 页面，或者一个 Servlet 包含到本页面中。它类似于 JSP 中的<jsp:include>标签。

A. include 标签　　B. file 标签

C. url 标签　　D. text 标签

(7) 在 Struts 2 的数据标签中的__________标签可以在页面中生成一个 Debug 超链接，单击这个超链接，可以查看到 ValueStack 和 ActionContext 中所有的对象。

A. text 标签　　B. i18n 标签
C. debug 标签　　D. bug 标签

(8) Struts 2 提供了 3 种方式可以创建一个新的主题，下面正确的是__________。

A. 重新创建一个全新的主题　　B. 引入其他框架主题
C. 结合其他框架主题编写一个新主题　　D. 删除现有主题的一部分得到新主题

(9) 下面属于表单标签的标签有__________。

A. form 标签　　B. iterator 标签　　C. a 标签
D. component 标签　　E. debug 标签　　F. url 标签

(10) 下面我们使用__________标签来输出一个 HTML 单行文本输入控件，相当于 HTML 代码：<input type="text" .../>。

A. textfield 标签　　B. text 标签
C. textarea 标签　　D. label 标签

(11) Struts 2 中的非表单标签中的__________标签可以用于使用自定义的组件。(单选)C

A. a 标签　　B. actionmessage 标签
C. component 标签　　D. actionerror 标签

三、上机练习

上机练习：选择自己喜欢的图书。

练习要求：本实例首先创建一个 JSP 页面，随意取名如：select_book.jsp。在该页面中使用<s:select>标签生成一个下拉选择框，并指定它的 list 属性加载一个图书信息列表，使用<s:optgroup>标签为下拉列选择框创建选项组，在下拉列选择框中分组显示图书信息。执行效果如图 8-10 所示。



图 8-10　选择自己喜欢的图书



第 9 章　轻松实现文件上传和下载

内容摘要：

文件上传是 Web 应用经常需要面对的问题。大多数情况下，用户的请求参数是在表单域输入的字符串，但如果为表单元素设置 enctype= “multipart/form-data” 属性，则提交表单时不再以字符串方式提交参数，而是以二进制编码的方式提交请求。此时直接通过 HttpServletRequest 的 getParameter()方法无法正常获取请求参数的值，可以通过二进制流来获取请求内容——通过这种方式，就可以取得希望上传文件的内容，从而实现文件的上传。

上面介绍的文件上传方式是全手工的文件上传机制，这种方式编程麻烦，而且需要全手动控制二进制流，过程繁琐。Struts 2 对此提供了相应的解决方案。

在一些网络系统中，需要隐藏下载文件的真实地址，或者将下载的数据存放在数据库中，那么可以通过编程来实现文件的下载，还可以对下载文件添加访问控制。

本章将介绍如何在 Struts 2 中实现文件的上传和下载。

学习目标：

- 掌握如何在 Struts 2 中实现文件上传。
- 掌握如何过滤文件上传的类型和大小。
- 掌握如何在 Struts 2 中实现文件下载。
- 熟练同时上传多个文件的步骤。

9.1　文件上传的原理

在网络上遇到许多朋友询问文件上传的问题，他们的问题或许千奇百怪，但主要原因都是因为不了解文件上传的原理。本节将从文件上传的底层机制讲起，希望帮助读者彻底解决文件上传的问题。

**视频教学**：光盘/videos/09/form.avi　　　　长度：6 分钟

在早期的 HTML 中，表单不能实现文件的上传，这多少限制了一些网页的功能。1995 年 11 月发布的 RFC1867 规范(即 HTML 中基于表单的文件上传)对表单做了扩展，增加了一个表单元素(<input type="file">)。如果在表单中使用了这个元素，浏览器在解析表单时，会自动生成一个输入框和一个按钮。输入框可供用户填写本地文件的文件名和路径，按钮可以让浏览器打开一个文件选择框供用户选择文件。这就是上传文件的起源。

1．表单元素的enctype属性

大多数情况下，无需设置表单元素的 enctype 属性，只需设置表单的 action 属性和 method 属性即可。其中 action 属性指定了表单提交到的 URL，method 属性指定是以 POST 方式还是以 GET 方式提交请求。

表单的 enctype 属性指定的是表单数据的编码方法，该属性有如下三个值。

```
application/x-www-form-urlencoded
```

这是默认的编码方式，它只处理表单域里的 value 属性值，采用这种编码方式的表单会将表单域的值处理成 URL 编码方式。

```
multipart/form-data
```

这种编码方式会以二进制流的方式来处理表单数据，这种编码方式会把文件域指定文件的内容也封装到请求参数里。

```
text/plain
```

该编码方式在表单的 action 属性为 mailto:URL 形式时比较方便，这种方式主要使用于直接通过表单发送邮件的方式。

下面以一个简单的 HTML 页面为例，来介绍 enctype 属性为 application/x-www-form-urlencoded 和 multipart/form-data 时的差别。

1)　表单的 enctype 为 application/x-www-form-urlencoded

创建一个简单的表单输入页面，代码片段如下所示。

```
<form action="form_success.jsp" method="post"
enctype="application/x-www-form-urlencoded">
        上传文件：<input type="file" name="file"><br>
        请求参数：<input type="text" name="paramNum"><br>
```

```
        <input type="submit" value=" 提交 " align="center" id="submit">
</form>
```

上面表单的 enctype 属性为 application/x-www-form-urlencoded，这是 enctype 属性的默认值。如果不指定该属性值，系统默认以 application/x-www-form-urlencoded 作为该属性的值。该页面的运行效果如图 9-1 所示。



图 9-1　包含文本框和文件域的表单

由于本应用主要用于分析表单元素的 enctype 属性，所以直接使用 JSP 页面来处理该请求。正如上面页面代码所实现的，该请求提交到 form_success.jsp 页面，该页面的代码内容如下。

```
<%@ page language="java" import="java.io.*" pageEncoding="gbk"%>
<%
        //获取 HTTP 请求的输入流
        InputStream is=request.getInputStream();
        //以 HTTP 请求输入流建立一个 BufferedReader 对象
        BufferedReader br=new BufferedReader(new InputStreamReader(is));
        //读取 HTTP 请求内容
        String buffer=null;
        while((buffer=br.readLine())!=null)
        {
            //在页面中显示读取到的请求参数
            out.println(buffer+"<br>");
        }
%>
```

显然，上面的处理页面直接通过二进制流来处理该 HTTP 请求——这是一种更底层的处理方式。当通过 HttpServletRequest 的 getParameter 方法来获取请求参数时，实际上是 Web 服务器替我们处理了这种底层的二进制流，并将二进制流转换成对应的请求参数值。

如果在如图 9-1 所示页面的文件上传域中选中要上传的图片，并在下面的输入框中输入“这是一棵树”，然后单击“提交“按钮，将会看到如图 9-2 所示的页面。



图 9-2　以application/x-www-form-urlencoded方式提交请求

从图 9-2 可以看出，即使通过底层的二进制输入流，一样可以读到该请求的内容：一个普通的字符串，整个普通字符串包含了三个请求参数，即 file、paramNum 和 submit。



即使是表单元素里的“提交”按钮，表单一样将其当成一个表单域，一样转换成一个请求参数。

大部分时候，程序中直接通过 HttpServletRequest 的 getParameter 方法即可获得正确的请求参数，而底层的二进制流处理和使用 URLDecoder 处理请求参数，都由 Web 服务器完成了。

因此，如果将 form_success.jsp 页面的代码改为如下简单形式。

```
<%@ page language="java" pageEncoding="gbk"%>
<%
        //设置 HttpServletRequest 使用 GBK 的编码方式
        request.setCharacterEncoding("GBK");
        //直接在页面上输出两个请求参数值
        out.println("paramNum 请求参数的值为
"+request.getParameter("paramNum")+"<br>");
        out.println("file 请求参数的值为
"+request.getParameter("file")+"<br>");
%>
```

此时再次运行程序，将看到如图 9-3 所示的页面。



图 9-3　直接获取请求参数的值

以上非常详细地介绍了当 enctype 属性为“application/x-www-form-urlencoded”时的作用。显然，如果设置该表单的 enctype 为“application/x-www-form-urlencoded”，则无法实现文件上传，为了实现文件上传，必须设置 enctype 属性为“multipart/form-data”。

2)　表单的 enctype 为 multipart/form-data

将表单提交页面中的 enctype 属性设置为“multipart/form-data”，并把 form_success.jsp 页面的代码改为前面直接使用二进制流处理的代码，该请求将会把文件域里浏览到的文件内容作为请求参数发送。显然，此时无法直接通过 request.getParameter()方法来获取请求参数。



一旦设置了表单的 enctype= “multipart/form-data” 属性，就将无法通过 HttpServletRequest 对象的 getParameter()方法取得请求参数。

2．手动上传

通过底层的二进制流来取得上传的文件内容，并将该文件内容放到 Web 应用所在路径下，从而实现文件上传。

对于每个文件域而言，总会有包含“filename=“xxx””的字符串……可以根据这些规律来处理文件的上传，下面是处理文件上传的 JSP 页面代码。

```
<%@ page language="java" import="java.io.*" pageEncoding="gbk"%>
<%
//取得 HttpServletRequest 的 InputStream 输入流
            InputStream is=request.getInputStream();
            //以 InputStream 输入流为基础，建立一个 BufferedReader 对象
            BufferedReader br=new BufferedReader(new InputStreamReader(is));
            String buffer=null;
            //循环读取请求内容的每一行内容
            while((buffer=br.readLine())!=null){
                //如果读到的内容以-----------------------------开始，且以--结束，
                  表名已到请求内容尾
                if(buffer.endsWith("--")  &&
buffer.startsWith("-----------------------------"))
                {
                            //跳出循环
                            break;
                }
                //如果读到的内容以-----------------------------开始，表名开始了
                  一个表单域
                if(buffer.startsWith("-----------------------------"))
                {
                            //如果下一行内容中有 filename 字符串，表明这是一个文件域
                            if(br.readLine().indexOf("filename")>1)
                            {
                                //跳过两行，开始处理上传的文件内容
                                br.readLine();
                                br.readLine();
                                //以系统时间为文件名，创建一个新文件
                                File file=new
```

```
File(request.getRealPath("/")+System.currentTimeMillis());
                                //创建一个文件输出流
                                PrintStream ps=new PrintStream(new
FileOutputStream(file));
                                String content=null;
                                //接着开始读取文件内容
                                while((content=br.readLine())!=null)
                                {
                                    //如果读取的内容以
-----------------------------开始，表明开始了下一个表单域内容

    if(content.startsWith("-----------------------------"))
                                    {
                                        //跳出处理
                                        break;
                                    }
                                    //将读到的内容输出到文件中
                                    ps.println(content);
                                }
                                //关闭输出
                                ps.flush();
                                ps.close();
                            }
                        }
                    }
                    br.close();
%>
```

通过上面的 JSP 页面，可以将一个文件上传到 Web 应用的根路径下。值得注意的是，在这里使用的是 BufferedReader 字符流(字符流在处理二进制文件时会出现问题)，因此上面的上传文件仅能处理文本文件的上传。



上面的上传处理页面不是一个非常完善的上传处理，因为它只能处理文本文件的上传。

当然，如果将上面采用字符流处理文件上传的逻辑，改为以字节流来处理文件上传，则上传处理将可以处理任何文件的上传。

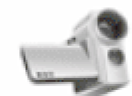
对于一个成熟的文件上传框架而言，它需要的逻辑非常简单：通过分析 HttpServletRequest 的二进制流，解析出二进制流中所包含的全部表单域，分析出每个表单域的类型(是文件域或普通表单域)，并允许开发者以简单的方式来取得文件域的内容字节、文件名和文件内容等信息，也可以取得其他表单域的值。

9.2　Struts 2 中的文件上传

Struts 2 并未提供自己的请求解析器。也就是说，Struts 2 不会自己去处理 multipart/form-data 的请求，它需要调用其他请求解析器，将 HTTP 请求中的表单域解析出来。但 Struts 2 在

原有的上传解析器基础上做了进一步封装，更进一步简化了文件的上传功能。

下面就介绍一下 Struts 2 对文件上传的支持和如何在 Struts 2 中实现文件的上传功能。

 视频教学：光盘/videos/09/struts2_upfile.avi　　长度：11 分钟

9.2.1　基础知识——Struts 2 对文件上传的支持

通过上一节的讲解，了解到上传文件的内容不能直接通过请求对象的 getParameter()方法来得到，需要以字节流的方式读取客户端提交的文件数据，并按照文件上传的格式对这些数据进行解析，从而获取上传文件的内容。那么，Struts 2 对文件上传的支持是怎样的呢？

Struts 2 本身没有提供解析上传文件内容的功能，它使用第三方的文件上传组件提供对文件上传的支持。Struts 2 默认使用的上传组件是 Apache 组织的 commons-fileupload 组件，该组件性能优异，而且支持任意大小文件的上传。Struts 2 还支持两种文件上传组件：pell 和 cos，可以通过配置 struts.multipart.parser 属性来切换 Struts 2 使用的上传组件。



对于 Struts 2 使用的上传组件，推荐读者使用 commons-fileupload 组件，该组件是目前最好的。

commons-fileupload 组件的下载网址是：http://commons.apache.org/fileupload/，本书使用的版本是 1.2.2。下载它的 Binary 压缩包(commons-fileupload-1.2.2-bin.zip)，解压缩后的目录中有两个子目录：lib 和 site。lib 目录下有一个 JAR 文件——commons-fileupload-1.2.2.jar 文件；site 目录中是 commons-fileupload 组件的文档，其中也包含了 API 文档。

commons-fileupload 组件从 1.1 版本开始依赖于 Apache 的另外一个项目：commons-io，它的下载网址是：http://commons.apache.org/io/，本书使用的版本是 2.0。下载它的 Binary 压缩包(commons-io-2.0-bin.zip)，解压缩后的目录有三个 JAR 文件，如下所示。

- commons-io-2.0.jar：这是 commons-io 的类库。
- commons-io-2.0-javadoc.jar：这是 commons-io 的 API 文档的压缩包。
- commons-io-2.0-sources.jar：这是 commons-io 的源代码的压缩包。

Struts 2 提供了一个文件上传拦截器：org.apache.struts2.interceptor.FileUploadInterceptor，它负责调用底层的文件上传组件解析文件内容，并为 Action 准备与上传文件相关的属性值。处理文件上传请求的 Action 必须提供特殊样式命名的属性，例如，假设表单中文件选择框(<input type="file" name="image">)的名字是 image，那么 Action 应该提供下列三个属性。

- image：java.io.File 类型，已上传文件的 File 对象。
- imageFileName：上传文件的文件名。
- imageContentType：上传文件的内容类型(MIME 类型)。

这三个属性的值会由 FileUploadInterceptor 拦截器准备。

9.2.2　实例描述

最近公司接了一个 OA(办公自动化)系统项目，既然是 OA 系统，文档管理模块必不可少。

具体要求就是能上传 txt、doc、jpg、gif……类型的文件。客户的要求很 “独特”，不仅要求要有“前台”还要有一个“后台”，本着客户就是上帝的宗旨，公司承诺做出一个完整的 OA 系统。

在此，我就给读者分享一下“前台”中的上传文档功能。

9.2.3 实例应用

【例 9-1】 OA 系统“前台”上传文档。

(1) 编辑上传文件的 Action 类(FileUploadAction.java)，在其内定义三个变量，分别代表上传文件的 file 对象、上传文件名、上传文件的 MIME 类型，并重写父类的 execute()方法，把上传的文件保存至 WEB-INF/UploadFiles 文件夹下。FileUploadAction 类的内容如下。

```
package com.struts2.actions;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Date;
import org.apache.struts2.ServletActionContext;
import com.opensymphony.xwork2.ActionSupport;
/**
 * 文件上传 Action 类
 * @author Administrator
 *
 */
public class FileUploadAction extends ActionSupport {
    //代表上传文件的 file 对象
    private File file;
    //上传文件名
    private String fileFileName;
    //上传文件的 MIME 类型
    private String fileContentType;

    //上传文件的描述信息
    private String description;
    //保存上传文件的目录，相对于 Web 应用程序的根路径，在 struts.xml 文件中配置
    private String uploadDir;
    @Override
    public String execute() throws Exception {
        String newFileName=null;
        //得到当前时间自 1990 年 1 月 1 日 0 时 0 分 0 秒开始流逝的毫秒数，将这个毫秒数作为
          上传文件的新文件名
        long now=new Date().getTime();
        //得到保存上传文件的目录的真实路径
```

```
        String
path=ServletActionContext.getServletContext().getRealPath(uploadDir);
        File dir=new File(path);
        //如果这个目录不存在，则创建它
        if(!dir.exists()){
            dir.mkdir();
        }
        int index=fileFileName.lastIndexOf('.');
        //判断上传文件名是否有扩展名，以时间截取为新的文件名
        if(index!=-1){
            newFileName=now+fileFileName.substring(index);
        }else{
            newFileName=Long.toString(now);
        }
        BufferedOutputStream bos=null;
        BufferedInputStream bis=null;
        //读取保存在临时目录下的上传文件，写入新的文件中
        try{
            FileInputStream fis=new FileInputStream(file);
            bis=new BufferedInputStream(fis);

            FileOutputStream fos=new FileOutputStream(new
File(dir,newFileName));
            bos=new BufferedOutputStream(fos);

            byte[] buf=new byte[4096];

            int len=-1;
            while((len=bis.read(buf))!=-1){
                bos.write(buf,0,len);
            }
        }finally{
            try{
                if(null!=bis){
                    bis.close();
                }
            }catch(IOException ex){
                ex.printStackTrace();
            }
            try{
                if(null!=bos){
                    bos.close();
                }
            }catch(IOException ex){
                ex.printStackTrace();
            }
        }
        return SUCCESS;
    }
    /*下面是上面所有属性的 set、get 方法，这里省略*/
}
```

(2) 在项目 src 目录下新建 struts-config 文件夹，在其内新建 fileupload.xml 文件，配置 FileUploadAction 类。配置如下代码。

```
<package name="upload" namespace="/upload" extends="fileupload">   ❶
        <action name="upload" class="com.struts2.actions.FileUploadAction">
            <result name="success">/success.jsp</result>
            <!--动态配置图片的存放位置-->
            <param name="uploadDir">/WEB-INF/UploadFiles</param>   ❷
        </action>
</package>
```

在❶处，读者看到配置 package 时，extends 的值为 fileupload，fileupload 是在 struts.xml 文件中的 package 的 name 值，在这个文件中，继承了 struts.xml 文件中的命名空间。

在❷处，在配置文件中设置保存上传文件的目录。在 Action 映射中，可以使用<param>元素来设置 Action 的属性值，由 staticParams 拦截器提供支持，这个拦截器已经包含在 defaultStack 拦截器栈中。



FileUploadInterceptor 拦截器已经在 struts-default.xml 文件中定义(拦截器名为 fileUpload)，并且已被包含在 defaultStack 拦截器栈中，所以不需要再单独配置对它的引用了。

(3) 把 fileupload.xml 文件引入 struts.xml 文件中，struts.xml 文件内容如下。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- 指定拦截器的 DTD 信息 -->
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <!-- 通过常量配置 Struts 2 所使用的解码集 -->
    <constant name="struts.i18n.encoding" value="gbk" />
    <constant name="struts.devMode" value="true" />
    <!--把 fileupload.xml 文件引入到本文件中-->
    <include file ="struts-config/fileupload.xml" />
    <package name="fileupload" namespace="/fileupload"
extends="struts-default">
    </package>
</struts>
```

(4) 在 WebRoot 下创建上传文件页面 fileupload.jsp。代码片段如下。

```
<%@taglib prefix="s" uri="/struts-tags"%>
<s:form action="upload/upload.action" method="post"
enctype="multipart/form-data">
        <s:file name="file" label="请选择上传的文件" />
        <s:textarea name="description" cols="50" rows="10" label="文件描述
"></s:textarea>
        <s:submit value=" 上  传  " align="center" cssClass="button"/>
</s:form>
```

(5)　继续创建上传成功页面 success.jsp 页面，在页面中输出上传文件的文件名、文件类型和文件描述内容。

9.2.4　运行结果

运行 fileupload.jsp 页面，上传一张图片。如图 9-4 所示。

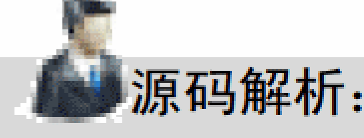
单击界面中的“上传”按钮，转至 SUCCESS 对应的上传成功页面(seccess.jsp)。如图 9-5 所示。



图 9-4　文件上传界面

图 9-5　上传成功界面

9.2.5　实例分析



源码解析：

上例子中，在 fileupload.xml 文件中以<param name="uploadDir">/WEB-INF/UploadFiles </param>动态的设置了 FileUploadAction 类中的 uploadDir 的值，其实也可以在 FileUploadAction 中初始化 uploadDir 的值。之所以在配置文件中动态设置是为了后期的维护，如果不想把上传的文件保存入这个文件夹路径下了，只需要修改一下 fileupload.xml 文件即可。

9.3　上传文件过滤

大多数时候，Web 应用不允许浏览者自由上传，尤其不允许上传可执行性文件(因为可能是病毒程序)。通常，可以允许浏览者上传图片、上传压缩文件等。除此之外，还必须对浏览者上传的文件大小进行限制。因此必须在文件上传中进行文件过滤。

这一节将为读者介绍如何对浏览者(用户)上传文件进行过滤，和对文件上传进行更多的控制。



视频教学：光盘/videos/09/ struts2_upfile.avi　　　　**长度：**11 分钟

9.3.1 基础知识——对文件上传进行更多的控制

对于控制上传文件的大小、格式，Struts 2 提供了一个文件上传的拦截器——FileUpload Interceptor 拦截器，通过配置该拦截器可以更轻松地实现文件过滤。

这个拦截器负责处理文件上传操作，它是默认的 defaultStack 拦截器栈的一员。即使你对拦截器一无所知，也可以轻松完成对上传文件的管理工作。不过，了解一下这个拦截器的工作原理可以更好地在 Struts 2 应用程序里使用文件上传功能。

FileUploadInterceptor 拦截器有两个重要的属性是开发者通常用到的。通过设置以下两个属性，可以对上传文件的长度和允许上传的内容类型进行限制。

- maximumSize：上传文件的最大长度(以字节为单位)，默认值大约是 2MB。
- allowedTypes：以逗号分隔的内容类型的列表(例如：text/html)，符合列表中的类型的文件，可以被传给 Action。如果没有指定这个参数，那么就是允许任何上传类型。

下面的动作对上传文件的最大长度和内容类型做出了限制——只允许用户上传长度大于 1 000 000 个字节的 JPEG、GIF 和 PNG 文件。

```
<action name="FileUpload" class="com.struts2.actions.FileUploadAction">
        <interceptor-ref name="fileUpload">
            <param name="maximumSize">1000000</param>
            <param name="allowedTypes">
                image/gif,image/jpeg,image/png
            </param>
        </interceptor-ref>
    </action>
```

fileUpload 拦截器的 maximumSize 参数只是设定 Action 能接受的文件的最大长度(在 Action 处理之前，文件已经上传到服务器上了)，而不是对上传文件的最大长度进行限制。

如果要对上传文件的最大长度进行限制，可以通过设置 struts.multipart.maxSize 属性来实现，该属性将直接影响 Struts 2 框架底层使用的 commons-fileupload 组件对文件的接受处理。

如果上传的文件长度大于 struts.multipart.maxSize 属性的值，那么底层的 commons-fileupload 组件将抛出 org.apache.commons.fileupload.FileUploadBase$SizeLimit Exceeded Exception 异常，上传文件拦截器捕获到该异常后，将直接把该异常的消息设置为 Action 级别的错误消息。如果在页面中使用<actionerror>标签，将看到如下的错误。

```
the request was rejected because its size(6249303) exceeds the configured
maximum(2097152)
```

编辑 struts.xml 文件，添加 struts.multipart.maxSize 属性的设置，如下所示。

```
<constant name="struts.multipart.maxSize" value="1000000"/>
```

在上传页面中可以用<s:actionerror/>标签调用，输出文件过滤失败后的错误提示信息。

如果用户上传的文件尺寸大于给定的最大长度或是其内容类型没被列在 allowedTypes 参数里，将会显示一条出错消息。与文件上传有关的出错消息已经在 struts-messages.properties 文件里预先定义好，这个文件被打包在 Struts 2 的系统 JAR 文件里。下面是这个文件的部分内容。

```
struts.message.error.uploading=Error uploading:{0}
struts.message.error.file.too.large=File too large:{0} "{1}" {2}
struts.message.error.content.type.not.allowed=Content-Type not
allowed:{0} "{1}" {2}
```

其中：

- struts.messages.error.uploading：文件不能上传的通用错误信息。
- struts.messages.error.file.too.large：上传的文件长度太大的错误信息。
- struts.messages.error.content.type.not.allowed：当上传的文件不匹配指定的内容类型时的错误信息。

如果需要覆盖这些默认的出错提示信息，就需要在 WEB-INF/classes/org/apache/struts2 子目录下创建一个名为 struts-messages.properties 的文件，然后使用同样的 key 把需要覆盖的默认信息替换为自己的东西。



如果创建了一个新的 struts-messages.properties 文件，Struts 2 将不再使用它自带的同名文件。因此，如果需要覆盖一条信息并继续使用其他默认出错提示信息的情况下，千万不要忘记把它们复制到新创建的属性文件中去。

9.3.2　实例描述

做开发者有一个令人很头痛的事，那就是当你把程序做好时，客户又要求修改。有的项目是改了数遍，以至于看到那个项目就心烦意乱。上次给客户做的 OA 系统中的上传文档功能，客户都运行了几天了，结果又说不行。问其原因是他们公司有人捣鬼，把非常大的文件上传上去了，结果导致服务器崩溃。当时他不用限制上传文件，很大的、多种格式的都能上传，遇到麻烦后，才要求修改项目。

这次，客户的需求是：只能上传图片，大小在 1000 字节以内。

9.3.3　实例应用

【例 9-2】 控制 OA 系统中上传文档大小和格式。

(1)　在上一节的案例(OA 系统“前台”上传文档)基础上，修改 fileupload.xml 文件，为上传文件的 Action 类(FileUploadAction 类)配置 FileUploadInterceptor 拦截器，配置它的 maximumSize 值为 1000、allowedTypes 值为 image 类型。配置代码如下。

```
<action name="upload" class="com.struts2.actions.FileUploadAction">
            <interceptor-ref name="fileUpload">
                <param name="maximumSize">10000</param>
                <param name="allowedTypes">
                    image/gif,image/jpeg,image/png
                </param>
            </interceptor-ref>
            <!--为 FileUploadAction 配置 defaultStack 拦截器引用-->
```

```
        <interceptor-ref name="defaultStack"/>
        <!--配置文件过滤失败后要跳转的页面-->
        <result name="input">/fileupload.jsp</result>
        <result name="success">/success.jsp</result>
        <param name="uploadDir">/WEB-INF/UploadFiles</param>
    </action>
```



在配置上传文件 Action 时除了必须为 Action 配置名称为 input 的逻辑视图，还必须显示地为该 Action 配置拦截器引用。

(2) 在本项目的 src 下新建 org.apache.struts2 包，并在这个包下创建 struts-messages.properties 文件，修改上传过滤失败后的提示错误信息内容。struts-messages.properties 的内容如下。

```
#改变文件类型不允许的提示信息
struts.messages.error.content.type.not.allowed=您上传的文件类型只能是图片文件！请重新选择！
#改变上传文件太大的提示信息
struts.messages.error.file.too.large=您要上传的文件太大，请重新选择！
```

(3) 修改 fileupload.jsp 页面内容，输出错误提示信息，代码如下。

```
    <!-- 使用红色的前景色输出错误信息 -->
    <div style="color:red">
        <s:fielderror/>
    </div>
```

(4) 在 fileupload.xml 文件中添加下面代码限制上传文件的最大长度。

```
<constant name="struts.multipart.maxSize" value="1000000"/>
```

9.3.4 运行结果

再次运行 fileupload.jsp 页面。当上传的图片过滤失败时，提示错误信息，如图 9-6 所示。



图 9-6 上传图片过滤失败提示错误信息

9.3.5　实例分析



源码解析：

上例子中，在项目下新创建的 org.apache.struts2 包下创建了 struts-messages.properties 文件。这样，在运行项目时，Struts 2 将不会再使用它自带的同名的文件，因此当文件过滤失败后，会读取自己新创建的 struts-message.properties 文件中的内容，提示错误信息。

9.4　同时上传多个文件

同时上传多个文件在 Web 开发中也是必不可少的功能。在 Struts 2 中实现多文件上传也不是很难，可以将多个<s:file .../>绑定到 Action 的 File 数组或 File 列表中。

下面将介绍一下如何实现同时上传多个文件。



视频教学： 光盘/videos/09/upmanyfile1.avi　　长度：7 分钟
光盘/videos/09/ upmanyfile2.avi　　长度：6 分钟

9.4.1　基础知识——同时上传多个文件

在前面讲到上传文件的动作类中必须有三个属性，这三个属性的名字必须是以下格式。

```
[inputName]File
[inputName]FileName
[inputName]ContentType
```

这里的“[inputName]”是 JSP 页面上的 file 标签的名字。如果是上传单个文件，[inputName]File 属性的类型就是 java.io.File，它代表被上传的文件；[inputName]FileName 属性和[inputName]ContentType 属性的类型是 String，它们分别代表被上传文件的文件名和内容类型。

如果上传多个文件，可以使用数组或 java.uitl.List。例如，下面的属性分别是 File 数组和 String 数组。

```
private File[] image;
private String[] imageFileName;
private String[] imageContentType;
```

如果使用 List，就必须把这三个属性都赋值为一个空白的列表。

```
private List<File> image=new ArrayList<File>();
private List<String> imageFileName=new ArrayList<String>();
private List<String> imageContentType=new ArrayList<String>();
```

然后循环遍历 image 数组或者集合实现单个文件上传。

9.4.2 实例描述

前几天，一个朋友对我说，他的项目经理要求他给一个客户做政府部门的系统，其中有一个功能的实现令他很是头痛。因此他特意咨询我。

他给我说了一大堆有关这个难住他的功能细节后，我才明白，原来就是一个多文件上传，让他没辙了。大概给他说了一下多文件上传的功能思路后，他恍然大悟。

下面的案例就实现了多文件上传功能，希望读者不要像我朋友那样遇到棘手的问题没辙……

9.4.3 实例应用

【例 9-3】 实现多文件上传功能。

(1) 编辑多文件上传的动作类——MultiFileUploadAction.java，在其内部定义三个数组变量，分别代表多个上传文件、多个上传文件的文件名和多个上传文件的 MIME 类型。重写父类的 execute()方法，循环遍历多个文件，实现多文件上传功能。内容如下。

```
package com.struts2.actions;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Date;
import org.apache.struts2.ServletActionContext;
import com.opensymphony.xwork2.ActionSupport;
/**
 * 文件上传 Action 类
 * @author Administrator
 *
 */
public class MultiFileUploadAction extends ActionSupport {
    //使用 File 对象数组，接收多个上传文件
    private File[] file;
    //使用数组保存多个上传文件的文件名
    private String[] fileFileName;
    //使用数组保存多个上传文件的 MIME 类型
    private String[] fileContentType;
    //上传文件的描述信息
    private String description;
    //保存上传文件的目录，相对于 Web 应用程序的根路径，在 struts.xml 文件中配置
    private String uploadDir;
    @Override
    public String execute() throws Exception {
        String newFileName=null;
```

```
        //循环处理多个上传文件
        for(int i=0;i<file.length;i++){
            //得到当前时间自 1990 年 1 月 1 日 0 时 0 分 0 秒开始流逝的毫秒数，将这个毫秒数作
              为上传文件的新文件名
            long now=new Date().getTime();
            //得到保存上传文件的目录的真实路径
            String
path=ServletActionContext.getServletContext().getRealPath(uploadDir);
            File dir=new File(path);
            //如果这个目录不存在，则创建它
            if(!dir.exists()){
                dir.mkdir();
            }
            int index=fileFileName[i].lastIndexOf('.');
            //判断上传文件名是否有扩展名，以时间截取为新的文件名
            if(index!=-1){
                newFileName=now+fileFileName[i].substring(index);
            }else{
                newFileName=Long.toString(now);
            }

            BufferedOutputStream bos=null;
            BufferedInputStream bis=null;
            //读取保存在临时目录下的上传文件，写入新的文件中
            try{
                FileInputStream fis=new FileInputStream(file[i]);
                bis=new BufferedInputStream(fis);

                FileOutputStream fos=new FileOutputStream(new
File(dir,newFileName));
                bos=new BufferedOutputStream(fos);

                byte[] buf=new byte[4096];

                int len=-1;
                while((len=bis.read(buf))!=-1){
                    bos.write(buf,0,len);
                }
            }finally{
                try{
                    if(null!=bis){
                        bis.close();
                    }
                }catch(IOException ex){
                    ex.printStackTrace();
                }

                try{
                    if(null!=bos){
                        bos.close();
                    }
```

```
                }catch(IOException ex){
                    ex.printStackTrace();
                }
            }
        }


        return SUCCESS;
    }
    /*下面是上面所有属性的 set、get 方法，这里省略*/
}
```



实现多文件上传可以使用上面所用的数组之外，还可以用 java.util.Lst，至于用 List 实现，这里不再详述，因为它和上面使用数组实现多文件上传的 Action 代码几乎相同，只是使用 List<File>代替了上面的 File[]，并使用 List<String>代替了 String[]，也就是说基本上是一样的，读者可以试着用 java.util.List 来实现多文件上传功能。

(2) 在 struts-config 文件夹下新建 multifileupload.xml 文件，配置 MultiFileUploadAction 类。完整内容如下。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- 指定拦截器的 DTD 信息 -->
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
    <package name="multiupload" namespace="/multiupload"
extends="fileupload">
        <action name="multiupload"
class="com.struts2.actions.MultiFileUploadAction">
            <result name="input">/multifileupload.jsp</result>
            <result name="success">/success.jsp</result>
            <!--动态配置 uploadDir 属性的值-->
            <param name="uploadDir">/WEB-INF/UploadFiles</param>
        </action>
    </package>
</struts>
```

(3) 把刚创建的 multifileupload.xml 文件引入 struts.xml 文件。

(4) 在 WebRoot 下新建 multifileupload.jsp 页面，代码如下。

```
<%@ page language="java" import="java.util.*" pageEncoding="gbk"%>
<%@taglib prefix="s" uri="/struts-tags"  %>
<%
    String path = request.getContextPath();
    String basePath = request.getScheme() + "://"
            + request.getServerName() + ":" + request.getServerPort()
            + path + "/";
```

```
%>
<base href="<%=basePath%>">
<s:form action="multiupload/multiupload.action" method="post"
enctype="multipart/form-data">
        <s:file name="file" label="请选择上传的文件一"/>
        <s:file name="file" label="请选择上传的文件二"/>
        <s:file name="file" label="请选择上传的文件三"/>
        <s:submit value="上传" align="center" cssClass="button"/>
</s:form>
```

(5)　继续在 WebRoot 下新建上传文件成功页面 success.jsp。

9.4.4　运行结果

运行 multifileupload.jsp 页面，出现如图 9-7 所示的界面。



图 9-7　多文件上传界面

9.4.5　实例分析



源码解析：

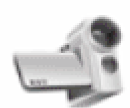
上例子中，通过查看 Action 代码，可以发现通过数组来实现上传多个文件不会比上传单个文件复杂多少，只需要提供三个数组属性分别封装多个上传文件的文件名、文件类型和文件内容即可。在处理文件上传的 execute()方法种，只需要遍历每个需要上传的文件，并将每个文件逐次写入服务器即可。Struts 2 系统负责将三个文件域对应的文件内容、文件名和文件类型，对应放入 Action 实例中的文件内容数组、文件名和文件类型数组中——这个过程无需开发者关心，这个过程也有力地证明了 Struts 2 框架的简单易用。

9.5 文 件 下 载

Struts 2 提供了 stream 结果类型，该结果类型是专门用于支持文件下载功能的。指定 stream 结果类型时，需要指定一个 inputName 参数，该参数指定了一个输入流，这个输入流是被下载文件的入口。

通过 Struts 2 的文件下载支持，允许系统控制浏览者下载文件的权限，包括实现文件名为非西欧字符的文件下载。

下面将介绍一下在 Struts 2 中的文件下载功能的实现。

 **视频教学：** 光盘/videos/09/filedown.avi　　长度：6 分钟

9.5.1 基础知识——Struts 2 对文件下载的支持

可能很多读者会认为，文件下载很简单，直接在页面上给出一个超链接，该链接的 href 属性等于要下载文件的文件名，不就可以实现文件下载了吗？大多数时候用这种方式可以实现文件下载，但如果该文件的文件名为中文文件名，则会导致下载失败；应用程序需要在用户下载之前进行进一步检查，比如判断用户是否有足够权限来下载该文件等，并且通过超链接下载文件，会暴露下载文件的真实地址，不利于对资源进行安全保护，而且利用超链接下载文件，服务器端的文件只能存放在 Web 应用程序所在的目录下。

利用程序编码实现文件下载，可以增加安全访问控制，对经过授权认证的用户提供下载；还可以从任意位置提供下载的数据，可以将文件放到 Web 应用程序以外的目录中，也可以将文件保存到数据库中。

利用程序实现文件下载非常简单，只需要按照如下的方式设置三个报头域就可以了。

```
Content-Type:application/x-msdownload
Content-Disposition:attachment;filename=downloadfile
Content-Length:filesize
```

浏览器再接收到上述的报头信息后，就会弹出“文件下载”对话框，询问用户是否将文件保存到本地硬盘。

1. Struts 2 对文件下载的支持

Struts 2 通过 org.apache.struts2.dispatcher.StreamResult 结果类型来支持文件下载，使得原来编写就简单的下载程序变得更加简单了。StreamResult 结果类型利用 HttpServletResponse 对象返回的 ServletOutputStream 对象向客户端输出下载文件的二进制数据，它有下列参数。

```
contentType
```

发送给 Web 浏览器的数据流的 MIME 类型(默认是 text/plain)，即下载文件的内容类型。

```
contentLength
```

数据流的长度，以字节为单位(浏览器显示一个进度栏)，即下载文件的长度。

```
contentDispostion
```

用于控制文件下载的一些信息，可选的设置包括：inline、filename="下载文件名"和 attachment。其中，filename="下载文件名"中，filename 指定下载的文件名；inline 表示下载文件在本页面内部打开；attachment 表示弹出“文件下载”对话框。不过，这也不是绝对的，对于浏览器能够显示的下载文件是这样的，但是对于浏览器不支持的下载类型，即使使用 inline 选项，仍然会弹出“文件下载”对话框。contentDispostion 的默认值是 inline。

```
inputName
```

Action 中用来下载文件的属性的名字，该属性的类型是 InputStream。默认值是 inputStream。

```
bufferSize
```

文件数据从输入复制到输出的缓冲区的大小，默认为 1024 字节。

2. 配置文件下载Action

配置文件下载的 Action 与配置普通的 Action 并没有太大的不同，需要在配置普通的 Action 的基础之上，再加上额外的 download 的拦截器引用。除此之外，关键是需要配置一个类型为 stream 的结果，配置时需要指定如下四个属性。

- contentType：指定被下载文件的文件类型。
- inputName：指定被下载文件的入口输入流。
- contentDisposition：指定下载的文件名。
- bufferSize：指定下载文件时的缓冲大小。

因为 stream 结果类型的逻辑视图是返回给客户端一个输入流，因此无需指定 location 属性。



配置 stream 类型的结果时，无需指定实际显示的物理资源，也无需指定 location 属性，只需要指定 inputName 属性即可，该属性指向被下载文件。

下面是配置该下载所用的 Action 类的配置文件。

```
            <result name="success" type="stream">
                <!-- 指定下载文件的内容类型 -->
                <param name="contentType">image/jpg</param>
                <!-- 指定下载文件的文件位置，它的默认值是 inputStream，如果在 Action
中用于读取下载文件内容的属性名是 inputName，那么这里可以省略这个参数的配置 -->
                <param name="inputName">targetFile</param>
                <param name="contentDisposition">
                    filename=tree.jpg
                </param>
                <!-- 指定下载文件的缓冲大小 -->
                <param name="bufferSize">4096</param>
            </result>
```

如果通过上面的 Struts 2 提供文件下载支持来实现文件下载，就可以实现包含中文文件名的文件下载了。

9.5.2 实例描述

一般情况下，有文件上传的存在就有文件下载的存在。上次给客户做的 OA 系统，也存在文件下载功能。当用户把上传的文件提交给上级领导后，领导可以下载相应的图片进行查看操作，以供审核。

以下的实例为读者展示文件下载功能的实现。

9.5.3 实例应用

【例 9-4】 OA 系统的文件下载功能。

(1) Struts 2 的文件下载 Action 与普通的 Action 并没有太大的不同，仅仅是该 Action 需要提供一个返回 InputStream 流的方法，该输入流代表了被下载文件的入口。该 Action 类的代码如下。

```
package com.struts2.actions;
import java.io.InputStream;
import org.apache.struts2.ServletActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class FileDownloadAction extends ActionSupport {
    private String inputPath;//下载文件的路径，在 struts.xml 文件中配置
    /**
     * 下载用的 Action 应该返回一个 InputStream 实例
     * 给方法对应在 result 里的 inputName 属性值为 targetFile
     * @return
     */
    public InputStream getTargetFile() throws Exception{
        return
ServletActionContext.getServletContext().getResourceAsStream(inputPath);
    }
    /**
     * 依赖注入该属性值的 setter 方法
     * @param inputPath
     */
    public void setInputPath(String inputPath) {
        this.inputPath = inputPath;
    }
    /**
     * 处理用户请求的 execute()方法，该方法返回 success 字符串
     */
    public String execute() throws Exception {
        return SUCCESS;
    }
}
```

从上面的 Action 中看出，该 Action 中包含了一个 getTargetFile()方法，该方法返回一个 InputStream 输入流，这个输入流返回的是下载目标文件的入口。该方法的方法名为 getTargetFile，表明该 Action 有一个 targetFile 属性类返回下载文件。

(2) 在 struts-config 文件夹下新建 filedownload.xml 文件，配置文件下载 Action 类——FileDownloadAction 类，配置如下。

```
<struts>
    <package name="filedownload" namespace="/filedownload"
extends="fileupload">
        <action name="filedownload"
class="com.struts2.actions.FileDownloadAction">
            <!-- 指定下载文件的路径，该路径相对于 Web 应用程序所在的目录 -->
            <param
name="inputPath">/WEB-INF/UploadFiles/1288346027796.jpg</param>
            <!-- 使用 StreamResult 结果类型 -->
            <result name="success" type="stream">
                <!-- 指定下载文件的内容类型 -->
                <param name="contentType">image/jpg</param>
                <!-- 指定下载文件的文件位置，它的默认值是 inputStream，如果在 Action
中用于读取下载文件内容的属性名是 inputName，那么这里可以省略这个参数的配置 -->
                <param name="inputName">targetFile</param>
                <param name="contentDisposition">
                    filename=tree.jpg
                </param>
                <!-- 指定下载文件的缓冲大小 -->
                <param name="bufferSize">4096</param>
            </result>
        </action>
    </package>
</struts>
```

(3) 把 filedownload.xml 文件引入 struts.xml 文件中。

(4) 创建 filedownload.jsp 页面，即下载文件页面，代码如下。

```
<%@taglib prefix="s" uri="/struts-tags"%>
<s:a href="filedownload/filedownload.action">下载图片</s:a>
```

9.5.4　运行结果

运行 filedownload.jsp 页面，出现如图 9-8 所示的界面。

单击“下载图片”超链接，出现如图 9-9 所示的界面。



图 9-8　文件下载界面

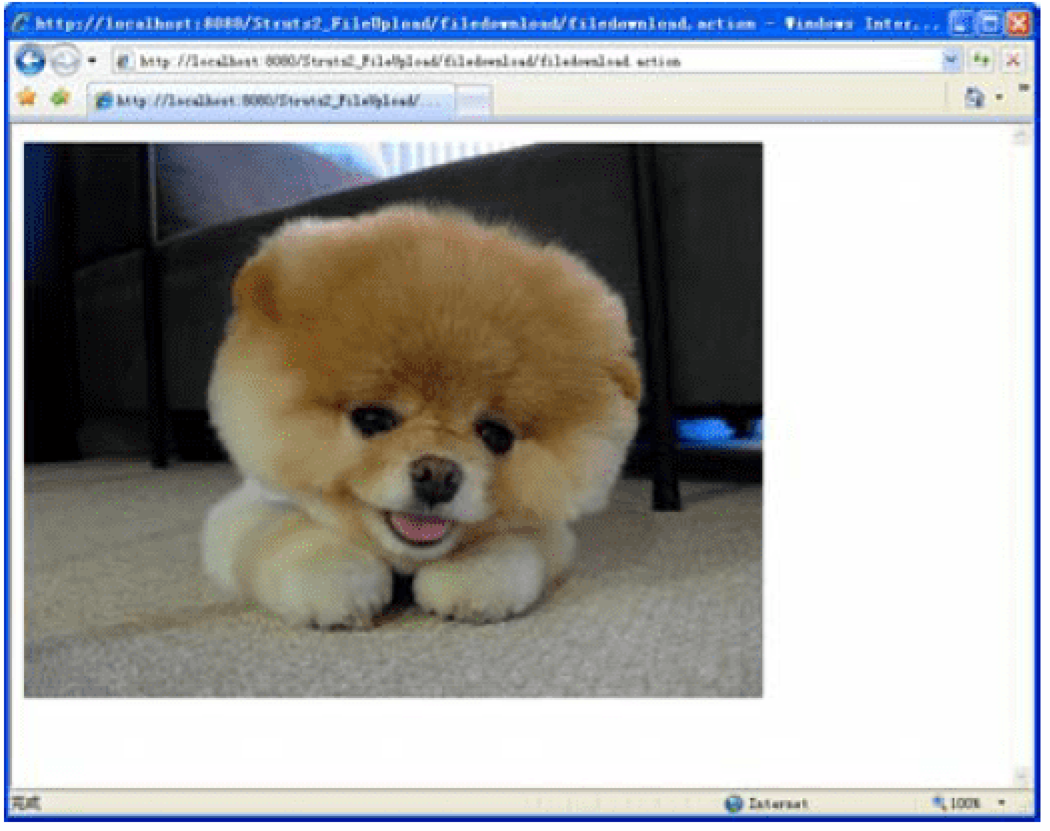


图 9-9　下载图片

9.5.5　实例分析



源码解析：

上个例子只是模拟了文件下载功能的实现，这里要下载的图片是固定的(图片路径是/WEB-INF/UploadFiles/1288346027796.jpg)，在实际开发中图片不可能是固定不变的，一般是动态加载的，因此可以在文件下载 Action 中获取数据库中要下载的文件并赋给 inputPath 属性。

contentDisposition 属性默认的是 inline，表示下载的文件内容在本页面中打开，因此，当单击“下载图片”超链接时会把要下载的图片——WEB/INF 下的 UploadFiles 文件夹下的 1288346027796.jpg 内容在下载文件页面中打开，即图 9-9 的效果。

9.6　常见问题解答

9.6.1　Struts 2 上传文件大小问题



Struts 2 上传文件大小问题！

网络课堂：http://bbs.itzcn.com/thread-10923-1-1.html

最近在学习 Struts 2 中的文件上传功能时，遇到一个棘手的问题。让我感到不解的是有时候上传文件不会出现错误，有时候却会错误，好像有人在捣鬼似的。这个异常就是 org.apache.commons.fileupload.FileUploadBase$SizeLimitExceededException。

从异常来看，好像与文件大小有关，于是看拦截器 FileUploadInterceptor.java 的源代码还是不明白是怎么回事？百度、Google 了许久，还是没找到问题的根本原因。

这是经常遇到的一个问题，由于 common-fileupload 组件默认最大支持上传文件的大小为

2M，所以在上传图片的时候，上传了大于 2M 的图片时，就会出现 org.apache.commons.fileupload.FileUploadBase$SizeLimitExceededException 异常。在控制台上并没有打印出这个异常，这个异常的发生导致了 fileUpload 拦截器没有机会执行，所以在上传 2M 以上的图片时，页面只是一闪而过，仍然停留在之前定义的 input 页面，什么错误提示都没有。

【解决办法】：就是在 struts.xml 配置文件的 struts 标签中添加如下代码。

```
<constant name="struts.multipart.maxSize" value="10485760"/>
```

这样就限制了上传文件的大小，把限制上传文件的大小定义的大一些，问题就解决了。

9.6.2　Struts 2 中，上传文件过大时，JSP页面也不显示错误



Struts 2 中，上传文件过大时，JSP 页面也不显示错误？

网络课堂：http://bbs.itzcn.com/thread-10924-1-1.html

在 Struts 2 中，当上传文件过大时，如何使 JSP 页面不显示错误信息？

【解决办法】：我们 Struts 2 本身提供了一个文件上传的拦截器，通过配置该拦截器可以更轻松地实现文件过滤。只需要在 Action 中配置该拦截器就 OK 了。当文件过滤失败后，会自动转向 input 逻辑视图，因此必须为该 Action 配置名为 input 的逻辑视图，除此之外还必须为配置 defaultStack 的拦截器的引用。配置文件如下。

```
<action name="upload" class="com.annlee.upload.UploadAction" >
    <!-- 配置 fileUpload 的拦截器 -->
    <interceptor-ref name="fileUpload">
        <!-- 配置允许上传的文件类型 -->
        <param name="allowedTypes">image/bmp,image/gif,image/jpg</param>
        <!-- 配置允许上传的文件大小 -->
        <param name="maximumSize">2000000</param>
    </interceptor-ref>
    <interceptor-ref name="defaultStack"></interceptor-ref>
    <param name="savePath">/</param>
    <result>/common/succ.jsp</result>
    <result name="input">/cos_fileupload/fileupload.jsp</result>
</action>
```

如果上传失败，系统会返回到原来的页面，要在原来的页面上加上以下错误提示代码：<s:fielderror/>这样，系统就会返回错误信息给用户，但是这时的提示是 Struts 2 自带的提示，非常不友好，配置国际化资源文件的以下两项，提示就会自动替换成自定义的 Struts 2 的提示，提示的关键字如下。

```
struts.messages.error.content.type.not.allowed
struts.messages.error.file.too.large
```

此外，如果用户上传失败的原因不是因为以上两项的原因，还有一个错误信息，它的关键字是如下。

```
struts.messages.error.uploading
```

9.6.3 Struts 2 上传文件后保存到我的项目文件夹中却是一个tmp文件



Struts 2 上传文件后保存到我的项目文件夹中却是一个 tmp 文件，怎么回事？
网络课堂：http://bbs.itzcn.com/thread-10928-1-1.html

问题是这样的：我上传一个图片为 a.jpg，结果到我的项目文件夹下却是 upload_82eela9_121ed4e4067_8000_0000003.tmp，找不到我的 a.jpg 了，这是什么原因？难道是代码有问题吗？下面是我的 Action 类代码片段。

```
    private File myFile;// 实际上传文件
    private String contentType;// 文件的内容类型
    private String fileName;// 上传文件名
    /*这里是上面三个属性的 set、get 方法，省略*/
    public void copyFile(File src, File target) {
        InputStream is = null;
        OutputStream os = null;
        byte[] number = new byte[BUFFER_SIZE];
        try {
            is = new BufferedInputStream(new FileInputStream(src),
BUFFER_SIZE);
            os = new BufferedOutputStream(new FileOutputStream(target),
            BUFFER_SIZE);
            while (is.read(number) > 0) {
                os.write(number);
            }
            os.close();
            is.close();
        } catch (Exception ex)
        {
            /*省略*/
        }
    }
    /**
    * 上传动作执行
    */
    public String loadPicture()
    {
        /**省略处理上传文件的操作*/
    }
```

【解决办法】：看了上面的代码，读者看出有什么不妥了吗？我来回答一下以上发生的错误原因所在。

首先在 Action 中的上传动作方法中编写 System.out.println("文件名字"+filename);代码，判断是否获取到了文件名(fileName 的是否为空)。

然后根据判断，确定原因：如果变量 fileName 为空，那么修改变量声明为如下所示。

```
private File myFile;//实际上传文件
private String myFileContentType;//文件的内容类型
private String myFileFileName;//上传文件名
```

修改的原因：Struts 2 上传文件是接收到前台的参数进行后台给变量赋值的，变量的名称的定义不是随便的。

9.6.4　Struts 2 上传中文文件名文件下载后编程乱码



Struts 2 上传中文文件名文件下载后编程乱码，怎么解决啊？
网络课堂：http://bbs.itzcn.com/thread-10929-1-1.html

【解决办法】：这个问题经常会遇到。其实方法很简单，就是转码。转码的方式有很多，经常用到的是下面这种方式。

```
String fileName = "中文文件名";//取得原中文文件名
fileName=new String(fileName.getBytes( "gb2312 "), "iso-8859-1 ");//处理中文乱码名
```

9.7　习　　题

一、填空题

(1) Struts 2 默认使用的上传组件是 Apache 组织的__________组件。

(2) Struts 2 提供了一个文件上传拦截器：__________，它负责调用底层的文件上传组件解析文件内容，并为 Action 准备与上传文件相关的属性值。

(3) 下面代码中，如果对只能上传图片进行控制，“__________”处应填写__________。

```
<interceptor-ref name="fileUpload">
    <param name="maximumSize">1000000</param>
    <param name="allowedTypes">
        __________
    </param>
</interceptor-ref>
```

(4) 如果上传多个文件，可以使用数组或__________。

(5) StreamResult 结果类型中的__________参数表示下载文件的名字。

二、选择题

(1) 不是表单元素的 enctype 属性有 3 个值，分别是__________。

A. application/x-www-form-urlencoded　　B. multipart/form-date

C. multipart/form-data　　D. text/plain

(2) 处理文件上传请求的 Action 必须提供特殊样式命名的属性，例如，假设表单中文件选择框(<input type="file" name="upload">)的名字是 upload，那么 Action 应该提供下列三个属性。下面代码中哪组是正确的________。

A.
```
private File upload;
private String uploadFileName;
private String uploadContentType;
```

B.
```
private String upload;
private String uploadFileName;
private String uploadContentType;
```

C.
```
private String upload;
private File uploadFileName;
private String uploadContentType;
```

D.
```
private File upload;
private String filename;
private String contentType;
```

(3) FileUploadInterceptor 拦截器有两个重要的属性可以对上传文件的长度和允许上传的内容类型进行限制，这两个属性是________。

A. maximumSize 和 allowedTypes　　B. maxLength 和 Types

C. max 和 allowedTypes　　D. maximumSize 和 Types

(4) 如果你想覆盖 Struts 2 内部提供的上传失败错误信息，就需要在 WEB-INF/classes/org/apache/struts2 子目录下创建一个名为 struts-messages.properties 的文件，然后使用同样的 key 把你打算覆盖的默认消息替换为你自己的东西。下面哪些是在 struts-messages.properties 中不出现的？ ________

A. struts.message.error.uploading

B. struts.message.error.file.too.large

C. struts.message.error.content.type.not.allowed

D. struts.message.error.allows

三、上机练习

上机练习：把上传的图片下载，并显示在页面上。

要求：

(1) 实现多文件上传。在表单中设置三个 File 元素，同时上传 3 张图片，如图 9-10 所示。

(2) 单击“上传”按钮后，跳转至上传成功页面，并把上传的图片显示在页面上，如图 9-11 所示的界面。



图 9-10　多文件上传页面



图 9-11　文件上传成功界面

(3) 单击“图片”，实现下载功能，并在本页面中显示出要下载的图片内容。如图 9-12 所示。

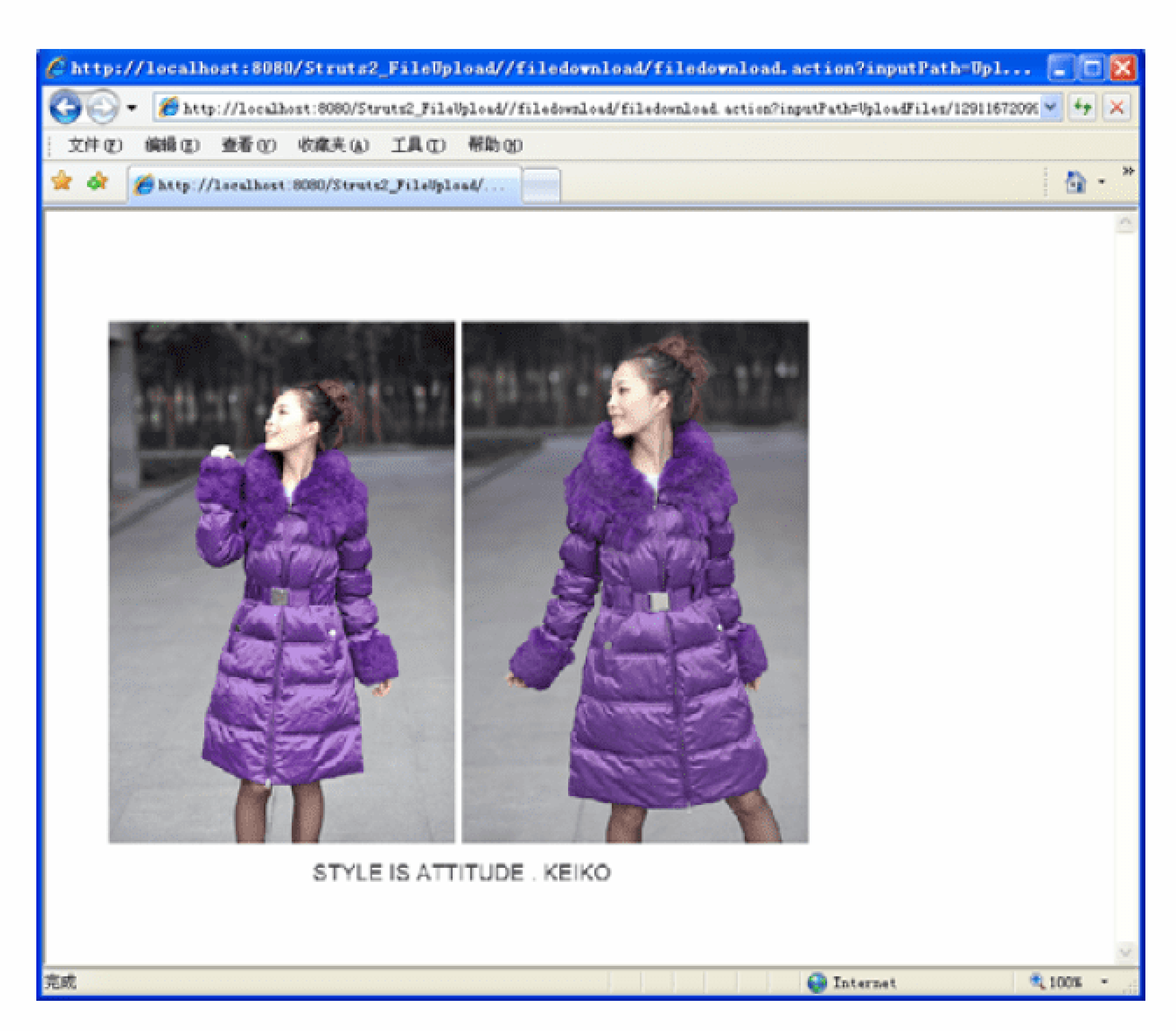


图 9-12　下载图片



第 10 章　避免表单重复提交和等待页面

内容摘要：

当用户在表单中填写完自己的信息内容之后，单击提交按钮后，可能没有看到自己想要的运行效果进而重复单击提交按钮，从而导致服务器接受到的信息是用户单击次数的条数信息内容，或者是注册时没有发现注册成功而重复单击导致数据库抛出异常。在实际应用中同样存在，由于用户没有及时看到相应的信息，导致重复提交的事时有发生。

为避免表单的重复提交可以在客户端通过 JavaScript 脚本实现，也可以在服务器端实现。本章介绍如何使用 Struts 2 避免表单重复提交，和检测表单是否重复提交机制。

学习目标：

- 防止表单重复提交的机制。
- 掌握 TokenIntercepter 拦截器的使用。
- 掌握 TokenSessionStoreInterceptor 拦截器使用。
- 使用 ExecuteAndWaitInterceptor 拦截器向用户显示等待页面。

10.1 避免表单重复提交

如果应用程序没有处理表单的重复提交现象，用户提交的“注册”、或者上传的文件等，将会被发送两次或者多次，从而导致在数据库中添加多条记录。

如何避免这种情况呢？具体实现时，可以在客户端通过 JavaScripte 脚本，或者在服务器端实现。例如，如果在 Action 中进行处理，读者很容易想到利用 session，在用户单击提交按钮时，判断该请求是否被提交过。

视频教学： 光盘/videos/10/token1.avi　　长度：10 分钟
光盘/videos/10/token.avi　　长度：10 分钟
光盘/videos/10/ tokenSession.avi　　长度：5 分钟

10.1.1 基础知识——token标签的作用

如何在生成页面时生成 token 字段呢？Struts 2 框架提供了 token 标签。例如，在 form 表单中，定义<s:token />语句，这样就在页面文件中添加 token 标签。在运行程序请求该页面文件时，在运行页面的源文件中，可以看到如下形式的代码。

```
<input type="hidden" name="struts.token"
value="QAP2MPXRBP9SPRF2JVESUUHRDQ04Z51W" />
```



token 标签根据每次请求的内容，将生成一个唯一性的隐藏字段 value 值，字段长度为 32 字节。

token 标签通过这个隐藏表单域，实现在服务器端避免表单重复提交，其实现原理如下。

(1) 服务器端在处理客户端请求时，创建一个 session 对象和一个 token 值(命名为 token1)，也称为令牌值。然后将 token1 作为隐藏表单域的值，随处理结果一起发送到客户端，同时将 token1 保存到 session 中。

(2) 服务器端在处理得到的请求之前，将请求中的 token1 与保存在当前用户 session 中的值进行比较，检查这两个值是否匹配。

(3) 如果相等，表示用户是第一次提交该表单，则清除 session 中的 token1，然后执行数据处理操作，同时产生一个新的 token 值(命名为 token2)，保存到 session 中，当用户重新访问提交数据页面时，将新产生的 token2 作为隐藏输入域的值。

(4) 如果用户退回到刚才的提交页面并再次提交，客户端传过来的 token 值，是 token1，而服务器端的 token 值已经为 token2，这两个 token 值不相等，将不再对用户的请求进行提交，从而有效地防止了重复提交的发生。



token 标签必须与 TokenInterceptor、TokenSessionStoreInterceptor 或者 ExecuteAndWaitInterceptor 等配合使用，这三个拦截器都能对 token 标签进行处理。

10.1.2　基础知识——使用TokenInterceptor

在本节中，将学习 token 标签和 token 拦截器的应用和实现避免表单的重复提交。token 标签定义在 JSP 页面文件中，token 拦截器配置在 struts.xml 文件中。

在此使用 org.apache.struts2.interceptor.TokenInterceptor.TokenInterceptor 拦截器防止用户重复提交的操作。当表单提交时，TokenIntorceptor 拦截器截取请求，获取标准的 struts.token.name 请求，得到保存令牌值的请求参数名，然后在根据这个请求参数获取令牌值内容，再根据令牌名，从 session 中取出<s:token>标签之前保存到 session 中的令牌值信息，紧接着对两个令牌值进行比较，如果 session 中的令牌值等于请求提交的令牌值，那么就删除 session 中的令牌值，并且调用 handleValidToken()方法，该方法直接调用 Action 的方法请求进行处理。如果两个值不相等，那么 handleInvalidToken()方法被调用。首先它会添加一个 Action 级别的错误消息到这个 Action 中，然后返回结果编码，从而跳过 Action 的执行。

TokenInterceptor 拦截器的处理过重，除了需要为 Acion 配置这个拦截器的引用外，同时还需要配置"invalid.token"结果映射，以便拦截器在表单重复提交时，将请求转向这个结果视图。

```
<%@ page language="java" import="java.util.*" pageEncoding="GB2312"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<h4>用户登录</h4>
<s:form action="login">
    <s:token/>                              //使用 token 标签
    <s:actionerror/>                        //使用 actionerror 标签
    <s:textfield name="userName" label="姓名"/>
    <s:password name="userPassword" label="密码"/>
    <s:submit value="登录"/>
</s:form>
```

上述内容的 from 表单中，使用<s:token/>语句添加 token 标签，使用 actionerror 标签输出错误信息。

10.1.3　基础知识——使用TokenSession StoreInterceptor

对表单重复提交后会向用户显示一条错误消息，在此需要使用 org.apache.struts2. interceptor. TokenSessionStoreInterceptor 拦截器。

TokenSessionStoreInterceptor 继承自 TokenInterceptor，并重写了 handleValidToken()方法和 handleInvalidToken()方法。handleValidToken()方法在 session 中保存了一个包含 ActionInvocation 实例和令牌对象，如果重复提交了相同的对象，那么 handleInvalidToken()方法从 session 中去除这个对象，然后根据保存的 Action 的状态重新输出结果页面，同时 Action 不会重复执行。如果不同的令牌或者令牌为 null，那么 handleInvalidToken()方法则返回 INVALID_TOKEN_CODE 结果信息，那么请求转向 invalid.token 结果码映射的视图。

tokenSession 拦截器扩展了 token 拦截器，但是 tokenSession 拦截器不会返回一个特殊的结果，也不会添加一个动作错误，只是阻断后面的提交，这样做的结果就是用户将看到同样的响

应，就好像只有一次提交。


tokenSession 拦截器的实现类是 TokenSessionStoreInterceptor。

如果使用 tokenSession 拦截器，只需要在前面的登录示例中，修改 struts.xml 文件中的代码，如下所示。

```
<package name="default" extends="struts-default">
    <action name="login" class="action.LoginAction">
        <interceptor-ref name="defaultStack" />
        <interceptor-ref name="tokenSession"/>         //配置 tokenSession 拦截器
        <result name="invalid.token">/login.jsp</result>
        <result name="success">/index.jsp</result>
    </action>
<package>
```

上述代码中，将 <interceptor-ref name="token"/> 修改为 <interceptor-ref name="token Session"/>，表示使用 tokenSession 拦截器。

10.1.4 实例描述

通过使用 org.apache.struts2.interceptor.TokenInterceptor.TokenInterceptor 拦截器来防止用户登录重复提交的操作，当用户重复单击登陆时进行信息提示，同时只向服务器发送一次请求。当用户单击登录时系统调用后台处理代码，然后转向至首页面，当用户重复提交，系统将给出消息提示，或者用户单击后退再次登录时，都会转向消息提示页面。

10.1.5 实例应用

【例 10-1】 用户登录表单重复提交。

(1) 首先借助于第四章第 6 节的登录案例进行完成。首先在工程目录内添加一个登录后的首页面和重复登录时提示消息页面。

(2) 接下来创建一个 LoginFormAction.java 的 Action 用来处理登录请求，返回登录后的首页面详细代码如下。

```
<!-- 省略部分代码 -->
public class LoginFormAction extends ActionSupport{
    public String execute()
    {
        return SUCCESS;
    }
}
<!-- 省略部分代码 -->
```

上述代码中，常见的 Action 继承与 ActionSupport 类，在该类中创建一个默认的方法 execute()方法用来处理登录请求然后返回 SUCCESS 页面。

(3) 配置 struts.xml 中的 LoginFormAction.java 的 Action 配置，设置 SUCCESS 转向页面，详细代码如下所示。

```
<!-- 省略部分代码 -->
    <action name="loginform" class="com.itzcn.action.LoginFormAction">
            <interceptor-ref name="token"/>
            <interceptor-ref name="defaultStack" />
            <result name="invalid.token">/error.jsp</result>
          <result name="success">/loginsuccess.jsp</result>
    </action>
<!-- 省略部分代码 -->
```

在<action>元素中，配置默认拦截器 defaultStack，<interceptor-ref name="token"/>配置 token 拦截器。在<result>元素中，配置 invalid.token 的返回结果。



由于拦截器按照 struts.xml 中所配置的顺序依次执行，为了更早的结束重复提交的处理，应该将 token 拦截器放在所有拦截器的前面。

(4) 最后登录页面提交表单 form 里面设置 token 的标签，详细代码如下所示。

```
<s:form action="loginform.action">
    <s:token/>                   //使用 token 标签
    <s:textfield name="userName" label="姓名"/>
    <s:password name="userPassword" label="密码"/>
    <s:submit value="登录"/>
</s:form>
```

上述代码中，在登录页面 form 提交值 loginform.action 时设置拦截器使用 token 标签。使用<s:actionerror/>标签输出错误信息，该标签配置在 error.jsp 错误消息提示页面。

10.1.6 运行结果

部署项目，启动 Tomcat，在地址栏里面输入 http://localhost:8080/ch10_1/login.action，运行结果如图 10-1 所示。



图 10-1 登录页面运行结果

单击“登录”按钮后，页面将会提交至 loginform.action 进行处理转向登录后的首页面如图 10-2 所示，当刷新登录后的首页面或者重复提交拦截器则会拦截跳转至消息提示页面如图 10-3，同时提示一些信息内容。



图 10-2　登录成功首页面

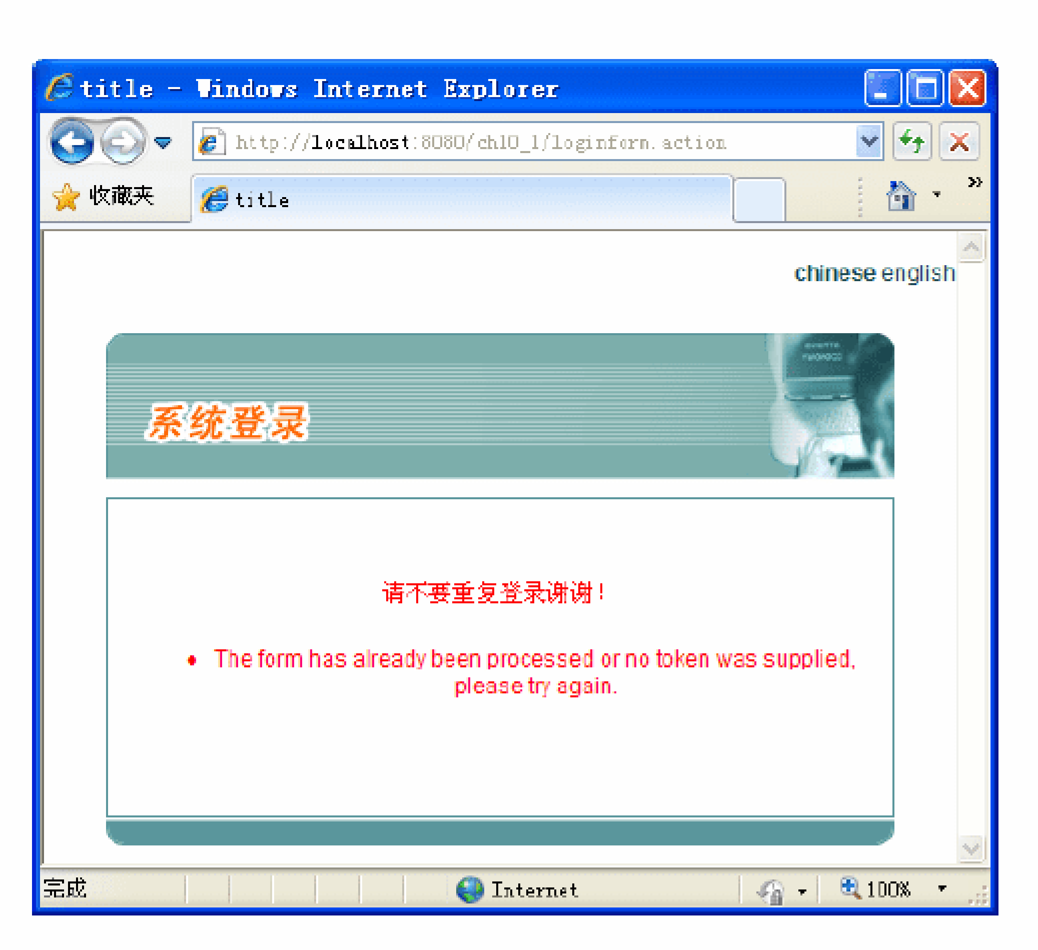
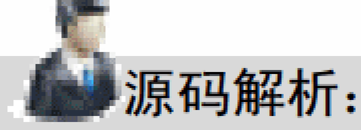


图 10-3　重复登录提示消息页面

10.1.7　实例分析



源码解析：

通过使用 TokenInterceptor 的拦截器获取用户请求判断，如果用户第一次表单提交访问 Action 则执行，如果表单第二次或者重复提交则转向 struts.xml 文件中配置<result>节点中的页面内。因此可以通过该拦截器来做到避免表单重复提交的问题，也可以避免用户在某些操作上出现的问题。

10.2　设置等待页面

有时候在处理某些操作或者请求的时候可能会耗费大量的时间，用户可能会不断的提交或者重复的刷新页面，这种情况下向用户显示一个等待的页面比采用防止用户重复提交的效果会更好一些。

Struts 2 也考虑到了这一点，它提供了 ExecuteAndWaitInterceptor 拦截器在用户提交表单后，向用户显示一个等待提交页面的提示消息，等待页面定时向服务器提交请求，以确定服务器获取的请求是否完成。如果请求已经完成，ExecuteAndWaitInterceptor 拦截器将把请求转向结果页面。

视频教学： 光盘/videos/10/execAndWait.avi　　长度：6 分钟
光盘/videos/10/execAndWait1.avi　　长度：5 分钟

10.2.1　基础知识——使用ExecuteAndWaitInterceptor

ExecuteAndWaitInterceptor 拦截器(简称：execAndWait)能够在用户请求提交之后执行一个耗费较长时间的 Action 在后台执行，而且向用户显示一个执行的进度信息。如果在请求一个 Action 执行的时间超过 10 分钟，它可以防止 HTTP 请求超时。

当用户提交一个表单请求时，ExecuteAndWaitInterceptor 拦截器将会创建一个新的线程来处理执行的 Action，同时向用户显示一个等待的消息内容，让用户知道请求处理的内容正在执行。等待页面中有一种自动刷新的功能，可以不断地通知浏览器在几秒钟发送一次请求，ExecuteAndWaitInterceptor 拦截器自动截取请求内容判断该请求是否被执行完毕，同时返回用户当前的请求信息，如果没有请求完毕则通知用户继续等待，如果请求执行完毕则转向结果页面。

ExecuteAndWaitInterceptor 是基于每个 session 工作的，也就是说同一个 Action 不可以在同一个 session 中运行一次或者多次。ExecuteAndWaitInterceptor 拦截器自身自带了一个请求等待的页面，它在 org/apache/struts2/interceptor/wait.ftl。这个等待页面非常简单，因此可以使用自己编写好的等待页面，为 Action 配置 wait 结果映射，转向编写好的等待页面。

ExecuteAndWaitInterceptor 拦截器已经在 struts-default.xml 文件中定义，但没有包含在 defaultStack 拦截器栈中，因此需要为 Action 配置引用这个拦截器。



defaultStack 拦截器必须配置为引用拦截器中的最后一个，因为它会停止后续的所有操作，在它之后的拦截器都不会被再次调用，ExecuteAndWaitInterceptor 拦截器创建的线程只会运行 Action，在 ExecuteAndWaitInterceptor 之后的所有拦截器都不会被调用执行。

ExecuteAndWaitInterceptor 初始化等待时间可以在服务器显示等待页面之前延迟一定的时间段。在延迟的时间内，ExecuteAndWaitInterceptor 拦截器每隔 100 毫秒都会自动检查后台线程是否执行完毕。如果因为某种原因该任务提前完成，那么等待页面就不会显示出来提示用户等待的消息内容。相反如果由于某种原因该任务需要耗费很大的时间段，则会显示出来等待的消息提示页面。ExecuteAndWaitInterceptor 拦截器有如下几个常用的属性参数。

- threadPriority：指定线程的优先级别，默认值为 Thread.NORM_PRIORITY。
- delaySleepInterval：该参数只能和 delay 参数使用，用来检查执行的程序是否执行完毕的间隔时间。默认值为 100 毫秒。
- delay：显示等待页面初始的等待延迟时间，默认是没等待延迟。



凡是被 ExecuteAndWaitInterceptor 拦截器拦截的 Action 将会单独创建一个 Action 来执行，因此这个 Action 不能使用 ActionContext，因为 ActionContext 是线程本地的。比如说访问了一个 session 的数据那么你就必须实现 SessionAware 接口而不能直接调用 ActionContext.getSession()方法。

10.2.2　实例描述

在这里同样使用登录系统来实现登录时转向等待页面，这次使用 ExecuteAndWait-

Interceptor 拦截器，在用户单击“登录”按钮提交登录信息后，服务器处理完毕之前向用户显示等待消息。

10.2.3 实例应用

【例 10-2】 用户登录自动显示登录页面。

(1) 在资源文件中添加等待提示消息内容。编辑 com.itzcn.action 包中的 LoginAction_zh_CN.properties 文件，添加等待提示消息，代码如下所示。

```
global.wait=您的请求正在处理，请稍后。<br/>如果页面没有自动重新加载，请<a href="{0}">
单击这里</a>
```

(2) 由于不希望使用到自带的等待页面，因此需要编写一个自己的等待页面，在 WebRoot\WEB-INF\freemarker 目录下创建一个 wait.html 文件，代码如下所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
    <head>
        <title>Please wait</title>
        <@s.url includeParams="all" id="url"/>
        <meta http-equiv="refresh" content=5;url=${url}>
    <meta http-equiv="Content-Type" content="text/html; charset=gb2312"
/><style type="text/css">
<!--
body,td,th {
    font-size: 12px;
    text-align: center;
}
-->
   </style></head>
    <body>
        <p><img src="images/20080318173010854.gif" /></p>
        <p><@s.text name="global.wait">
            <@s.param value="#url"/>
        </@s.text></p>
    </body>
</html>
```

在上述代码中，需要注意<@s.url>标签的 includeParams 属性必须使用，因为当等待页面刷新时，浏览器将会重新发送一次登录请求到 Action，如果没有用户登录消息，那么发送请求时将会出现空异常。

(3) 编写 struts.xml 文件，为 register action 配置引用 ExecuteAndWaitInterceptor 拦截器，同时配置 wait 结果映射。在上一节中引用到的 token 或者 tokenSession 拦截器，需要将这些拦

截器代码配置注释或者删除掉，因为 ExecuteAndWaitInterceptor 不能和 token 和 tokenSession 拦截器同时使用，详细代码如下所示。

```
<action name="loginform" class="com.itzcn.action.LoginFormAction">
<result name="wait" type="freemarker">
    /WEB-INF/freemarker/wait.html
</result>
        <interceptor-ref name="token"/>
        <interceptor-ref name="execAndWait">
            <param name="excludeMethods">default</param>
        </interceptor-ref>
        <result name="invalid.token">/error.jsp</result>
      <result name="success">/loginsuccess.jsp</result>
</action>
```

上述代码中，使用 result 节点配置 execAndWait 拦截器，获取拦截后转向节点中配置的路径内。



ExecuteAndWaitInterceptor 也是从 MethodFilterInterceptor 继承，所以可以配置排除和要拦截的方法列表。

同样还可以使用初始等待延迟，如果使用初始等待延迟，在配置 execAndWait 拦截器时，使用 delay 参数，代码如下所示。

```
<action name="login" class="action.LoginAction">
    <interceptor-ref name="defaultStack" />
    <interceptor-ref name="execAndWait">              //使用 execAndWait 拦截器
        <param name="delay">1000</param>              //配置 delay 参数
    </interceptor-ref>
    <result name="wait">/wait.jsp</result>            //等待页面 wait.jsp
    <result name="success">/index.jsp</result>
</action>
```

在上面的代码中，配置 execAndWait 拦截器，在该拦截器中使用 delay 参数，并设置参数值为 1000 毫秒。在运行程序时，服务器首先判断如果在 1000 毫秒之内，Action 能够处理完请求，将不显示等待页面，再根据 Action 的返回结果显示相应的视图。如果在 1000 毫秒内，Action 不能够处理完请求，则向用户显示 wait.jsp 页面。

10.2.4　运行结果

启动 Tomcat，在地址栏里面输入 http://localhost:8080/ch10_1/login.action，运行结果如图 10-4 所示。

当用户单击“登录”按钮之后，表单提交拦截器进行拦截转向等待页面，如图 10-5 所示。同时在后台执行提交登录操作。登录成功之后将看到图 10-6 所示的页面。



图 10-4　用户登录界面

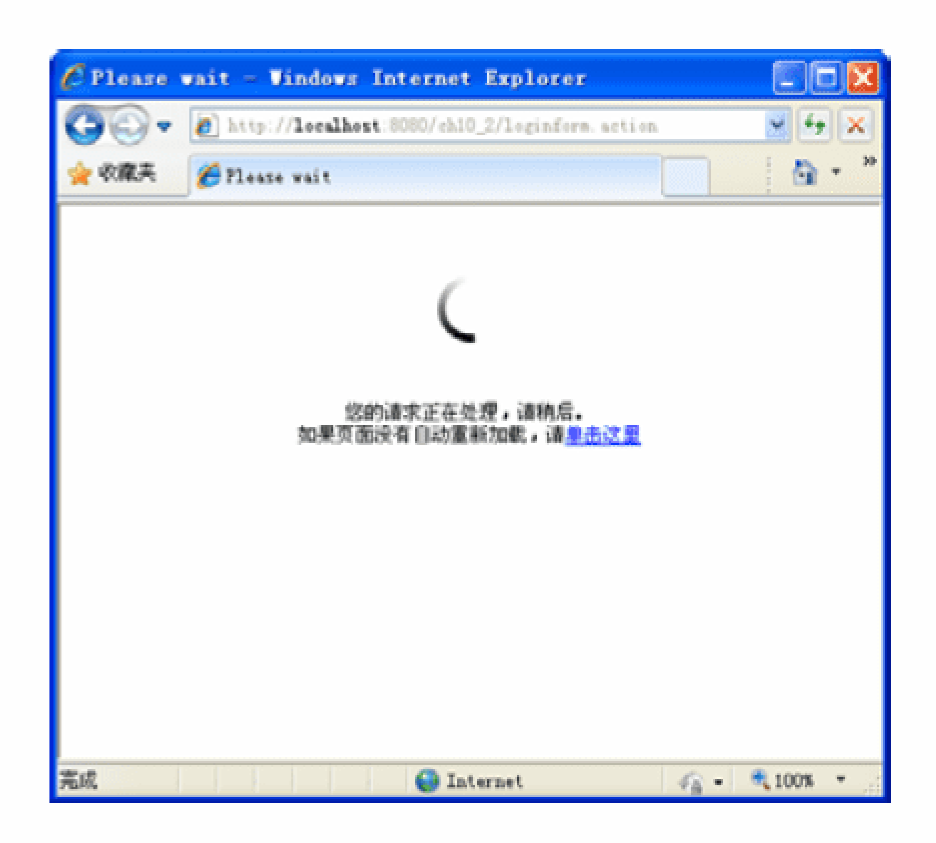


图 10-5　登录等待页面



图 10-6　登录成功页面

10.2.5　实例分析



源码解析：

通过以上实例可以将用户提交运行时间较长的程序设置后台运行，同时转向提交等待页面，这样大大地减少了用户不断提交的现象。而且在视觉上也比拦截重复提交的效果好，在该实例中，使用简单的 ExecuteAndWaitInterceptor 拦截器就完成了客户所需要的效果，即需要在 struts.xml 中找到需要拦截的 Action 即可。

10.3　常见问题解答



初次请求提交表单被拦截器拦截？

网络课堂：http://bbs.itzcn.com/thread-10932-1-1.html

在本章的第一节中讲解到了使用拦截器拦截表单的重复提交，当服务器端在处理得到的请求之前，将请求中的 token1 与保存在当前用户 session 中的值进行比较，检查这两个值是否匹

配。如果相等，表示用户是第一次提交该表单，清除 session 中的 token1，然后执行数据处理操作，同时产生一个新的 token 值，保存到 session 中。当用户重新访问提交数据页面时，将新产生的 token2 作为隐藏输入域的值。这样，如果用户退回到刚才的提交页面并再次提交，客户端传过来的 token 值，是 token1，而服务器端的 token 值已经为 token2，这两个 token 值不相等，将不再对用户的请求进行提交，从而有效地防止了重复提交的发生。但是万事总不是我们想象中那么完美，完成一个避免表单重复提交的实例之后，当要测试的时候，问题不请自来，因为第一次请求就会被拦截器拦截误认为重复提交。当程序出现这样的错误时首先检查配置信息是否正确，拦截器是否正常工作，最后就是在表单页面是否配置了标签。

```
<s:token/>
```

在运行程序请求该页面文件时，在运行页面的源文件中，可以看到如下形式的代码。

```
<input type="hidden" name="struts.token"
value="QAP2MPXRBP9SPRF2JVESUUHRDQ04Z51W" />
```

通过以上配置可以避免除此请求就被拦截器拦截。

10.4　习　　题

一、填空题

(1) Struts 2 框架利用__________机制，来解决 Web 应用中的重复提交问题。

(2) 本节学习到在 struts-default.xml 文件中，提供了对 TokenInterceptor、__________和 ExecuteAndWaitInterceptor 拦截器的配置。

(3) 由于拦截器按照 struts.xml 中所配置的顺序依次执行，为了更早的结束重复提交的处理，应该将__________拦截器放在所有拦截器的前面。

(4) __________拦截器扩展了 token 拦截器，但是 tokenSession 拦截器不会返回一个特殊的结果，也不会添加一个动作错误，只是阻断后面的提交。

(5) 5execAndWait 拦截器的主要参数有 threadPriority、delay 和__________ 三个常用参数。

二、选择题

(1) tokenSession 拦截器的实现类是__________。

A. TokenSessionStoreInterceptor　　B. TokenInterceptor
C. ExecuteAndWaitInterceptor　　D. 上述全不是

(2) 下述选项中那个属于 execAndWait 拦截器的参数__________。

A. class　　B. value　　C. delay　　D. priority

(3) 在 TokenInterceptor 拦截器中在页面中使用<s:token/>语句添加标签，使用__________输出错误信息。

A. param 标签　　B. text 标签
C. set 标签　　D. actionerror 标签

三、上机练习

上机练习：用户注册等待，防止用户重复提交。

要求： 制作一个会员注册系统如图 10-7 所示，当用户单击“注册”转向一个注册等待页面如图 10-8 所示，同时后台执行注册操作，注册成功则转向成功提示页面如图 10-9 所示。如果用户单击浏览器“后退”按钮重复提交能容则提示用户已经注册成功的消息如图 10-10 所示。



图 10-7　用户注册页面

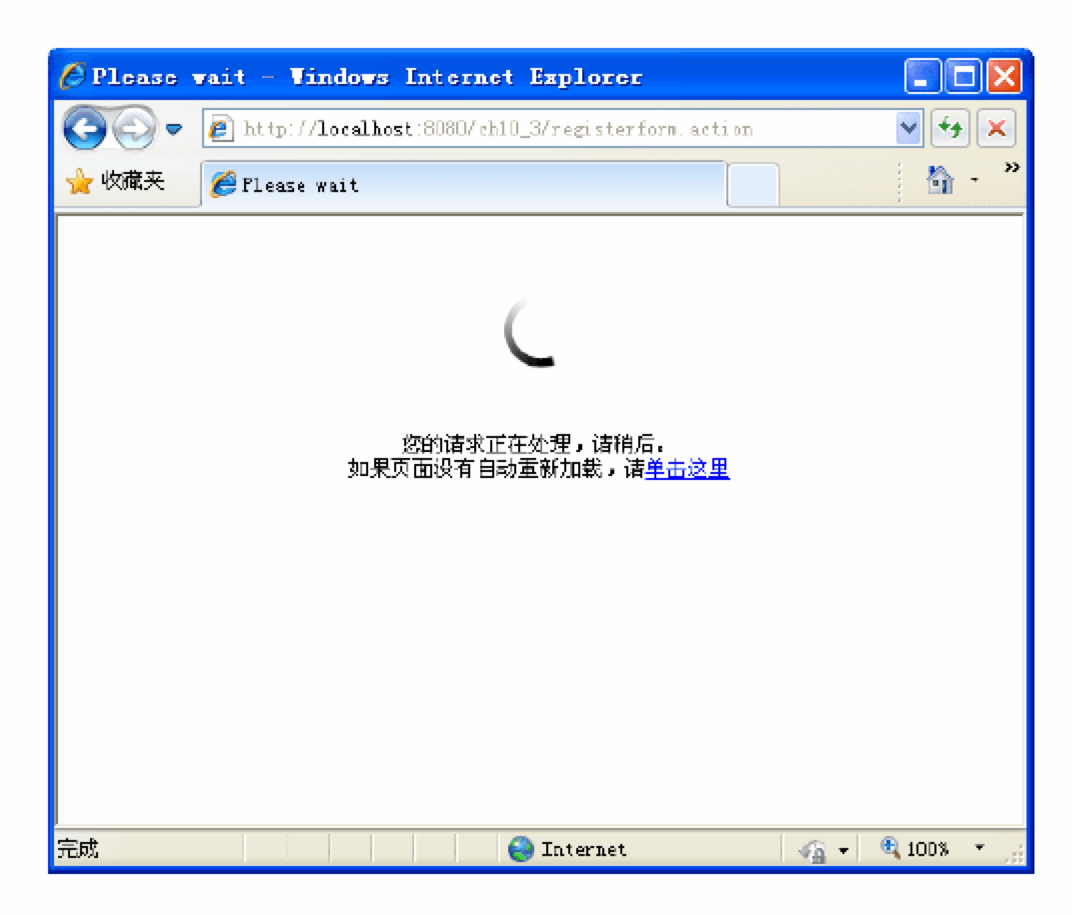


图 10-8　注册等待页面

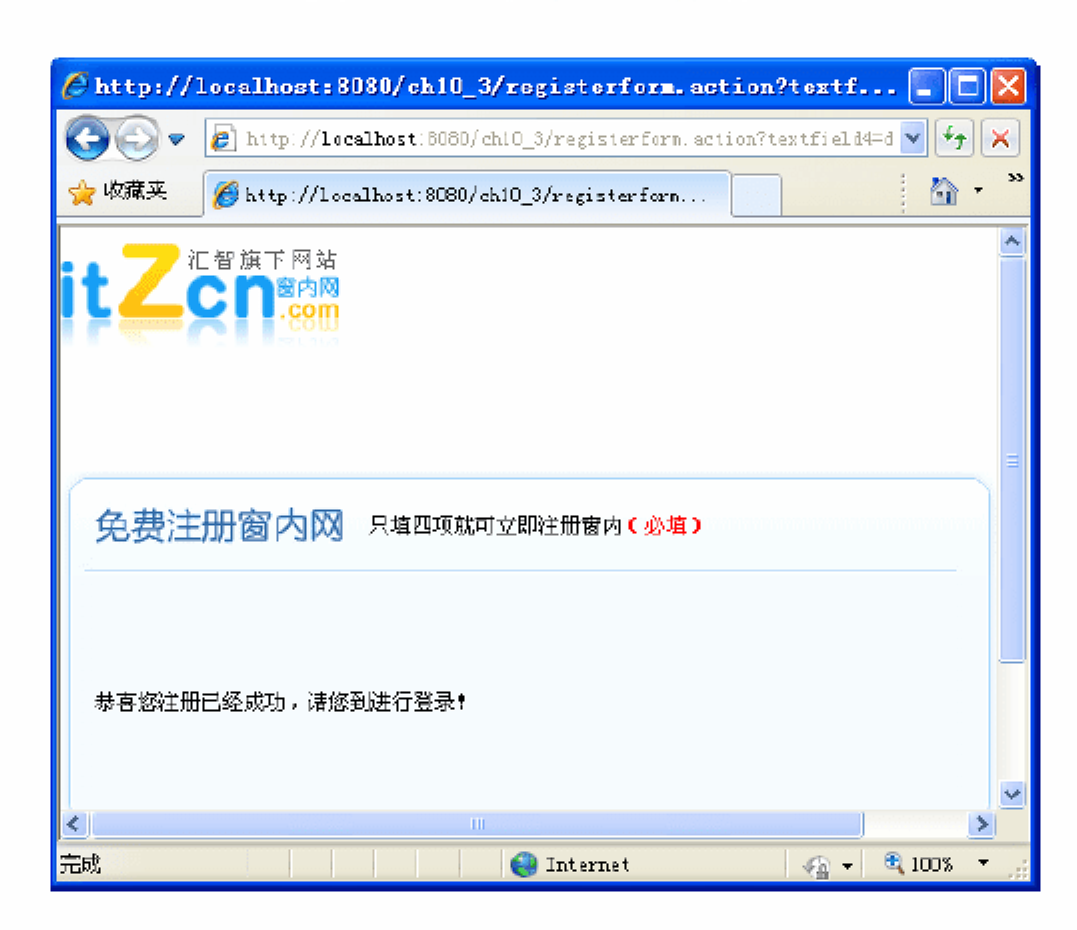


图 10-9　注册成功页面



图 10-10　重复注册提示页面



第 11 章 黄金搭档——Struts 2 集成 Spring 与 Hibernate

内容摘要：

Hibernate 是持久层的解决方案，Struts 2 是表现层的解决方案，Spring 是一个集成框架。Struts 2 通过插件的方式来集成 Spring，在添加用户程序中，使用 Spring 的依赖注入功能，为 SaveUserAction 注入它依赖的 UserDAO 对象。本章将介绍 Struts 2、Hibernate 和 Spring 的集成开发，简称 SSH2 组合。

学习目标：

- 理解 Hibernate 的作用。
- 掌握 Hibernate 的开发。
- 理解 Spring 的作用。
- 掌握 Struts 2 和 Hibernate 的集成开发。
- 掌握 Struts 2、Hibernate 和 Spring 的集成开发。

11.1 用户注册与登录

Java 是面向对象的语言，SQL 是结构化的语言，数据存储在关系型数据库中。在对象技术与关系技术结合应用的时候，会遇到一个问题。即，如何以面向对象的方式来操作关系型数据。这种需求催生了对象关系映射(ORM)技术，Hibernate 适逢其时地出现，得到了广大开发者的青睐，因为其快速的普及速度，成为最流行的对象关系映射框架。

 **视频教学：**光盘/videos/11/struts_hibernate.avi　　　　**长度：**13 分钟

11.1.1 基础知识——集成Hibernate

本节将简单介绍 Hibernate，和其基本操作，帮助读者快速了解 Hibernate 的开发。

1. Hibernate概述

Hibernate 是一个开放源代码的对象关系映射框架，它实现 ORM(Object-Relational Mapping，对象关系映射)，并对 JDBC 进行了轻量级的对象封装，使得 Java 开发者可以随心所欲地使用对象编程思维来操纵数据库(在具体的操作业务对象时，不需要和复杂的 SQL 语句打交道，只要像平时操作对象一样)。Hibernate 可以应用在任何使用 JDBC 的场合，既可以在 Java 的客户端程序使用，也可以在 Servlet/JSP 的 Web 应用中使用。最具革命意义的是，Hibernate 可以在应用 EJB 的 J2EE 架构中取代 CMP，完成数据持久化的重任。

2. Hibernate的优点

Hibernate 是 JDBC 的轻量级的对象封装。它是一个独立的对象持久层框架，和 App Server，和 EJB 没有什么必然的联系。Hibernate 可以用在任何 JDBC 可以使用的场合，例如，Java 应用程序的数据库访问代码，DAO 接口的实现类，甚至可以是 BMP 里面的访问数据库的代码。从这个意义上来说，Hibernate 和 EB 不是一个范畴的，也不存在非此即彼的关系。

Hibernate 是一个和 JDBC 密切关联的框架，所以 Hibernate 的兼容性和 JDBC 驱动以及数据库都有一定的关系，但是和使用它的 Java 程序，和 App Server 没有任何关系，也不存在兼容性问题。

Hibernate 不仅管理 Java 类到数据库表的映射(包括 Java 数据类型到 SQL 数据类型的映射)，还提供数据查询和获取数据的方法，可以大幅度减少开发时人工使用 SQL 和 JDBC 处理数据的时间。

Hibernate 能在众多的 ORM 框架中脱颖而出，是因为 Hibernate 与其他 ORM 框架对比具有如下优势。

- 开源和免费的 License，方便需要时研究源代码，改写源代码，进行功能定制。
- 轻量级封装，避免引入过多复杂的问题，调试容易，减轻开发者的负担。

- 具有可扩展性，API 开放。功能不够用时，自己编码进行扩展。
- 使开发者易于活跃，使产品有稳定的发展方向。

3. Hibernate的下载和安装

Hibernate 目前最新的版本是 3.6.0，本章使用的也是这个版本，下面按如下步骤来下载和安装 Hibernate。

(1) 登录到“http://nchc.dl.sourceforge.net/project/hibernate/hibernate3/3.6.0.Beta3/hibernate-distribution-3.6.0.Beta3-dist.zip”即可下载，下载 Hibernate 的二进制包。Windows 平台下载 zip 包，Linux 平台下载 tar 包。

(2) 解压下载的 hibernate-distribution-3.6.0.Beta3-dist.zip 包，在 hibernate-distribution-3.6.0.Beta3 路径下有一个 hibernate3.jar 压缩文件，该文件时 Hibernate 的核心类库文件。该路径下还有一个 lib 目录，该目录下存放了 Hibernate 编译和运行的第三方类库。

(3) 使用 Hibernate 应用可以复制 hibernate3.jar 核心类库文件，程序也会用到其他第三方类库可以根据需要到 lib 目录下进行复制使用。

(4) 如果开发的是 Web 应用，将上述文件复制到 WEB-INF/lib 路径下。

(5) 如果要在控制台编译应用 Hibernate API 类，只需将 hibernate3.jar 文件添加到 CLASSPATH 里。如果使用 Ant 工具，或者 Eclipse 等 IDE 工具，则不需要修改环境变量。

本书实例加载使用了 Hibernate 的如下 JAR 文件。

按照上述五个步骤完成之后，就可以使用 Hibernate 进行对象持久化操作了。

4. 使用Hibernate向数据库保存记录

使用 Hibernate 保存记录也就是将对象持久化，既然要持久化对象就一定要有对象。前面已经说过，Hibernate 是一个低侵入性的开源框架，完全可以采用一般的 Java 对象来作为持久化对象使用。一个一般的持久化对象类，代码如下所示。

```
public class Clothes {
    private int id;             //编号
    private String name;        //名称
    private String color;       //颜色
    private int price;          //价格
    //无参构造函数
    public Clothes(){

    }
    //带参构造函数
    public Clothes(String color, int id, String name, int price) {
        super();
        this.color = color;
        this.id = id;
        this.name = name;
        this.price = price;
    }
```

```
    //下面是 id、name、color 和 price 属性的 get 和 set 方法，在此省略了。
    //..........................................................

}
```

上述代码封装了一个简单的 Clothes 实体类，这个类与普通的 JavaBean 没有任何不同。这就是 Hibernate 的低侵入性的体现，不需要持久化类继承 Hibernate 的任何父类，或者实现任何接口。

仅仅使用上面定义的一个普通的 JavaBean 类是不能进行持久化操作的，Hibernate 采用了 XML 映射文件的方式来实现，使用 XML 文件对 Clothes 实体类进行映射配置，代码如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="pojo.Clothes">
        <id name="id">
            <generator class="native" />
        </id>
        <property name="name" />
        <property name="color"/>
        <property name="price"/>
    </class>
</hibernate-mapping>
```

上述代码中<hibernate-mapping>元素是 Hibernate 映射文件的根元素，这对于所有的映射文件都是相同的。在<hibernate-mapping>元素中添加一个子元素<class>来映射 Clothes 持久化类，使用<id>元素来为持久化类映射表主键字段，<property>元素为持久化类映射一般表字段。

通过上述的映射信息，我们理解了持久化类属性与数据库表列之间的对应关系，但不知道要连接哪个数据库以及连接数据库时所用的连接池、用户名和密码等详细信息。这些信息对于所有的持久化类都是通用的，可以通过 XML 文件来配置，代码如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<!--表明解析本 XML 文件的 DTD 文档位置，DTD 是 Document Type Definition 的缩写，
即文档类型的定义，XML 解析器使用 DTD 文档来检查 XML 文件的合法性
hibernate.sourceforge.net/hibernate-configuration-3.0dtd 可以在
Hibernate3.2.5 软件包中的 src\org\hibernate 目录中找到此文件-->
<!DOCTYPE hibernate-configuration PUBLIC
   "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<!--声明 Hibernate 配置文件的开始-->
<hibernate-configuration>
    <!--表明以下的配置是针对 session-factory 配置的，SessionFactory 是 Hibernate
    中的一个类，这个类主要负责保存 Hibernate 的配置信息以及对 Session 的操作-->
    <session-factory>
        <!--配置数据库的驱动程序，Hibernate 在连接数据库时，需要用到数据库的驱动
```

```
        程序-->
        <property name="hibernate.connection.driver_class">
            com.mysql.jdbc.Driver
        </property>
        <!--设置数据库的连接 url:jdbc:mysql://localhost:port/test,其中
        localhost 表示 mysql 服务器名称，此处为本机。port 代表 mysql 服务器的端口号,
        默认是 3306。test 是数据库名，这是你要连接的数据库名-->
        <property name="hibernate.connection.url">
            jdbc:mysql://localhost:3309/struts2_hibernate
        </property>
        <property name="connection.useUnicode">true</property>
        <property name="connection.characterEncoding">utf-8</property>
        <!--如果你的 mysql 服务器都是默认设置的，且装在本机器上，则也可以写成
            <property name="hibernate.connection.url">
                jdbc:mysql://localhost/test
            </property>或
            <property name="hibernate.connection.url">
                jdbc:mysql:///test
            </property>
        -->
        <!--连接数据库的用户名-->
        <property name="hibernate.connection.username">root</property>
        <!--连接数据库的密码-->
        <property name="hibernate.connection.password">123456</property>
        <!--hibernate.dialect 是 Hibernate 使用的数据库方言,就是要用 Hibernate 连接
        哪种类型的数据库服务器。-->
        <property name="dialect">
            org.hibernate.dialect.MySQLDialect
        </property>
        <!--hibernate.hbm2ddl.auto
        指定由 java 代码生成数据库脚本,进而生成具体的表结构的具体方式-->
        <property name="hbm2ddl.auto">update</property>
        <!--是否在后台显示 Hibernate 生成的查询数据库的 SQL 语句，开发时设置为 true，
        便于查询错误，程序运行时可以在 Eclipse 的控制台显示 Hibernate 执行的 Sql 语句。
        项目部署后可以设置为 false，提高运行效率-->
        <property name="show_sql">true</property>
        <!-- 开启二级缓存 -->
        <property
name="hibernate.cache.use_second_level_cache">true</property>
        <!-- 指定缓存产品提供商 -->
        <property name="hibernate.cache.provider_class">
        org.hibernate.cache.EhCacheProvider</property>
        <!-- 启用查询缓存 -->
        <property name="hibernate.cache.use_query_cache">true</property>
        <!--指定映射文件为"com/hibernate/dao/User.hbm.xml"-->
        <mapping resource="pojo/Clothes.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

上述代码配置一些连接数据库的驱动、URL、用户名、密码、数据库连接池的大小，等等。最后使用<mapping>元素引入了持久化类的映射文件。

一切工作准备就绪后，使用 Hibernate 向数据库中保存持久化类对象对应的记录。代码如下所示。

```
public class ClothesTest {
    public static void main(String[] args)throws Exception{
        //创建 Configuration 配置信息对象
        Configuration conf = new Configuration().configure();
        //创建 SessionFactory 对象
        SessionFactory sf = conf.buildSessionFactory();
        //创建 Session 对象
        Session session = sf.openSession();
        //开启事务
        Transaction tx = session.beginTransaction();
        //创建 Clothes 实体类对象，并为属性赋值
        Clothes clothes = new Clothes();
        clothes.setColor("红色");
        clothes.setName("羊毛大衣");
        clothes.setPrice(289);
        //保存实体类对象
        session.save(clothes);
        //提交事务
        tx.commit();
        //关闭 Session
        session.close();
    }
}
```

使用 session.save(clothes);即可持久化对象到数据库中，完全是面向对象操作。运行上述代码，在控制台上输出一条 insert sql 语句保存对象，效果如图 11-1 所示。

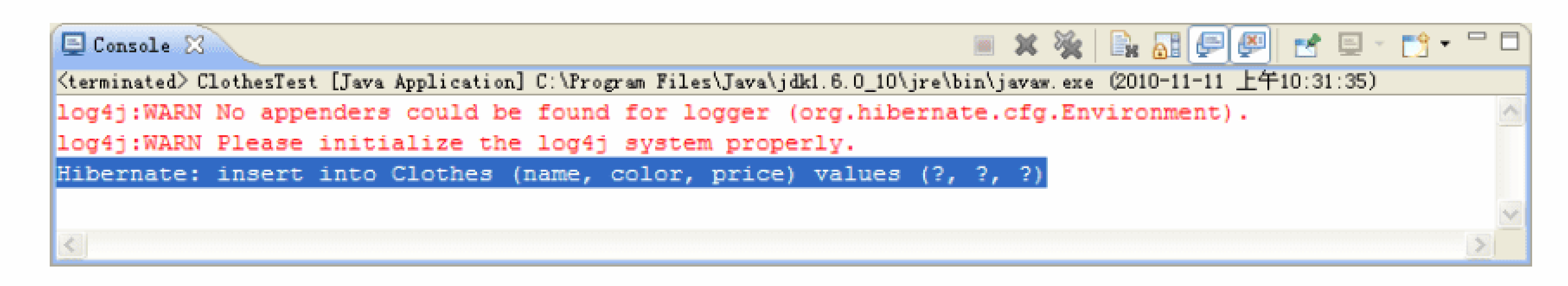


图 11-1　持久化Clothes对象

5. 使用Hibernate查询数据

Hibernate 不仅可以保存数据，也可以查询数据。它支持多种数据查询方式，包括最常用的 HQL 查询，也包括原生的 SQL 查询。Hibernate 也支持条件查询，条件查询需要使用以下三个类或接口：

- Criteria：代表一次条件查询。
- Criterion：代表一个查询条件。
- Restrictions：产生查询条件的工具类。

在条件查询中，Criteria 接口代表一次查询。该查询本身不具备任何的数据筛选功能，Session 调用 createCriteria(Class clazz)方法对某个持久化类创建条件查询实例。Criteria 常用的方法如表 11-1 所示。

表 11-1　Criteria常用的方法

方法名称	返回类型	作用说明
add(Criterion criterion)	Criteria	增加查询条件
addOrder(Order order)	Criteria	增加排序规则
list()	List	返回结果集
setFirstResult(int firstResult)	Criteria	设置查询返回的第一行记录
setMaxResults(int maxResults)	Criteria	设置查询返回的记录数

Criterion 接口代表一个查询条件，这个查询条件是由 Restrictions 负责产生，Restrictions 是一个用于产生查询条件的工具类，它的方法大部分都是静态方法，常用的方法如表 11-2 所示。

表 11-2　Restrictions常用的方法

方法名称	返回类型	作用说明
between(String propertyName,Object lo,Object hi)	Criterion	判断属性值在某个值范围之内
allEq(Map propertyNameValues)	Criterion	判断指定属性(参数 Map 的 key 值)和指定值(参数 Map 的 value)是否完全相等
ilike(String propertyName,Object value)	Criterion	判断属性值匹配某个字符串
ilike(String propertyName,String value,MatchMode matchMode)	Criterion	判断属性值匹配某个字符串，并确定匹配模式
in(String propertyName,Collection values)	Criterion	判断属性值在某个集合内
in(String propertyName,Object[] values)	Criterion	判断属性值是数组元素的其中之一
isEmpty(String propertyName)	Criterion	判断属性值是否为空
isNotEmpty(String propertyName)	Criterion	判读属性值是否不为空
isNotNull(String propertyName)	Criterion	判断属性值是否为空
isNull(String propertyName)	Criterion	判断属性值是否不为空
not(Criterion expression)	Criterion	对 Criterion 求否
sizeEq(String propertyName,int size)	Criterion	判断某个属性的元素个数是否与 size 相等
sqlRestriction(String sql)	Criterion	直接使用 SQL 语句作为筛选条件
sqlRestriction(String sql,Object[] values,Type[] types)	Criterion	直接使用带参数占位符的 SQL 语句作为条件，并指定多个参数值
sqlRestriction(String sql,Object value,Type type)	Criterion	直接使用带参数占位符的 SQL 语句作为条件，并指定参数值

使用 Hibernate 来实现一次条件查询，代码如下所示。

```
public static void main(String[] args) throws Exception{
    //创建一个 Configuration 对象
    Configuration conf = new Configuration().configure();
    //创建一个 SessionFactory 对象
    SessionFactory factory = conf.buildSessionFactory();
    //创建一个 Session 对象
    Session session = factory.openSession();
    //开启事务
    Transaction tx = session.beginTransaction();
    //获得一次查询条件对象
    Criteria crit = session.createCriteria(Clothes.class);
    //获取所有的 Clothes 实体对象
    List list = crit.list();
    //依次迭代取出 Clothes 对象并将对象名称与价格打印出来
    for (ListIterator iterator = list.listIterator();iterator.hasNext(); ) {
        Clothes clothes = (Clothes) iterator.next();
        System.out.println("衣服名称: " + clothes.getName());
        System.out.println("价格: " + clothes.getPrice());
    }
}
```

上述代码进行了一次条件查询，即可将所有的 Clothes 实体类对应的记录都查询出来，并依次迭代取出 Clothes 对象，输出显示衣服名称和价格，执行效果如图 11-2 所示。

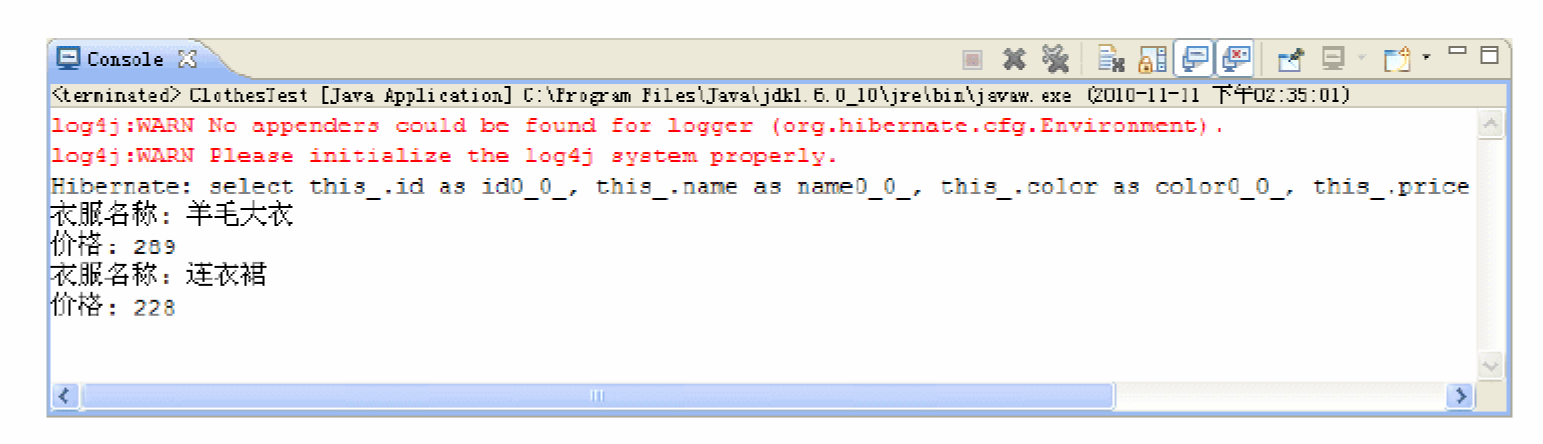


图 11-2 查询Clothes对象记录

6. Struts 2 与Hibernate的整合方案

在 Web 应用中，作为 MVC 的优秀框架，Struts 2 致力于与用户交互的层次。这样，可以将持久层交给 Hibernate 来管理。Struts 2 与 Hibernate 两个框架在应用中其实并没有直接的关联，它们不能直接交互。因此，要想整合 Struts 2 与 Hibernate，必须在两者之间插入其他解决方案。如图 11-3 所示 Struts 2 与 Hibernate 的整合架构方案。

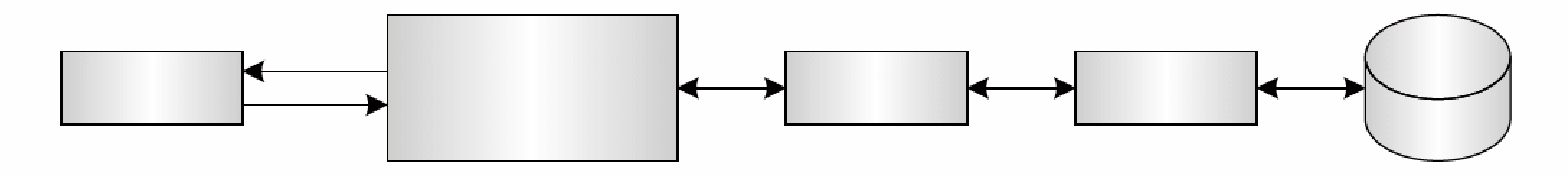


图 11-3 Struts 2 和Hibernate整合方案

上图中的中间层组件负责实现 Web 应用中的大部分业务操作。通常，中间层分为如下两个层。

- DAO 层：用于底层持久化实现，一个具体的 DAO 组件只能与特定的持久化技术耦合。
- 业务逻辑层：主要用于实现业务逻辑，这样可以避免与持久化技术耦合。



使用 Hibernate 框架可以代替 JDBC 技术，但是从图 11-3 可以看出，在 Hibernate 访问数据库时还是使用了 JDBC 技术。这是因为 Hibernate 的数据库访问建立在 JDBC 技术的基础上，只不过在应用中不需要显式使用 JDBC 编程而已。

11.1.2　实例描述

昨天完成一个项目后，闲暇之余，到论坛里看看别人发表的帖子。从发帖中发现许多人在学习 Struts 2、Hibernate 和 Spring 的整合。这三个框架确实是目前非常流行的框架，而且我刚做的一个项目也是使用 S2SH 三大框架做的。因为这三个框架有了新的认识，所以就给他们编写了一个例子，分享一下我的收获。本实例中的项目业务比较简单，便于读者理解，使用的是 Struts 2 和 Hibernate 两个主流框架做的。

11.1.3　实例应用

【例 11-1】 用户注册与登录。

(1) 新建一个 Web 项目 Struts2_11。在项目 src 目录下新建一个 pojo 包，在该包下新建一个用户实体类 User，声明 id、username、password、truename、sex、job 和 tip 属性，代码如下所示。

```
public class User {
    private int id;                 //编号
    private String username;        //用户名
    private String password;        //密码
    private String truename;        //真实姓名
    private String sex;             //性别
    private String job;     //职位
    private String tip;     //提示

    //下面是属性 id、username、password、truename、sex、job 和 tip 的 get 和 set 方法，
在此省略了。
    //……………………………………
}
```

(2) 在实体类 User 所在目录下，新建一个 User.hbm.xml 文件，在该文件中配置 User 实体类的映射数据库表的信息。代码如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
```

```
<hibernate-mapping>
<!-- 映射实体类 User -->
    <class name="pojo.User">
<!-- User 的 id 属性映射为 user 表的主键，native 表示为自增主键 -->
        <id name="id">
            <generator class="native" />
        </id>
<!-- 下面映射 User 的 username、password 等属性为 user 表的对应字段 -->
        <property name="username" column="username"/>
        <property name="password" column="password"/>
        <property name="truename" column="truename"/>
        <property name="sex" column="sex"/>
        <property name="job" column="job"/>
        <property name="tip" column="tip"/>
    </class>
</hibernate-mapping>
```

(3) 通过上述两步，持久层已经完成了。接下来在项目 src 目录下新建一个 dao 包，该包用于存放项目 dao 层组件，即本实例中的 UserDao。它是一个接口，定义两个方法，一个是 select()用于查询用户，另一个是 save()用于用户注册。代码如下所示。

```
package dao;
import java.util.Collection;
import pojo.User;
public interface UserDao {
    //添加一个用户，即可以用于用户注册
    public boolean save(User user);
    //查询用户名为 name 的用户
    public User select(String name);
}
```

(4) 新建一个 dao.impl 包，在该包下新建一个 UserDaoImpl 类来实现 dao 层的 UserDao 接口，代码如下所示。

```
package dao.impl;

import org.hibernate.Query;
import org.hibernate.Transaction;
import org.hibernate.Session;
import pojo.User;
import util.HibernateUtil;
import dao.UserDao;

public class UserDaoImpl implements UserDao {
    //实现查询用户名为 name 的用户
    public User select(String name) {
        User user = null;
        Session session = null;
        Transaction trans = null;
```

```
        try{
            session = HibernateUtil.getSession();
            trans = session.beginTransaction();
            Query query = session.createQuery("from User u where
u.username= ?" );
            query.setString(0, name);
            if(null != query.list() || !query.list().isEmpty()){
                user = (User) query.list().get(0);
            }
            trans.commit();
        }
        catch(Exception e){
            trans.rollback();
            e.printStackTrace();
        }finally{
            session.close();
        }
        return user;
    }
    //保存用户 user，用于实现用户注册
    public boolean save(User user){
        Session session = null;
        Transaction trans = null;
        boolean result = false;
        try{
            session = HibernateUtil.getSession();

            trans = session.beginTransaction();
            session.save(user);
            result = true;
            trans.commit();
        }
        catch(Exception e){
            trans.rollback();
            e.printStackTrace();
        }finally{
            session.close();
        }
        return result;
    }


}
```

(5)　在项目的 src 目录下新建一个 action 包，在该包下新建一个 UserAction，在该 Action 中定义一个 register()方法用于处理用户注册请求，重新继承于 ActionSupport 类的 execute()方法用户处理用户登录请求。代码如下所示。

```
package action;
```

```
import pojo.User;
import com.opensymphony.xwork2.Action;
import com.opensymphony.xwork2.ModelDriven;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ActionContext;
import dao.UserDao;
import dao.impl.UserDaoImpl;
public class UserAction extends ActionSupport implements ModelDriven<User>{
    //使用模型驱动
    private User user = new User();
    //实现 ModelDriven 接口必须实现的方法
    public User getModel(){
        return user;
    }
    //处理用户注册
    public String register(){
        UserDao userDao = new UserDaoImpl();
        boolean b = userDao.save(user);
        if(b){
            return "success";
        }else{
            getModel().setTip("注册失败，请重新注册！");
            return "error";
        }
    }
    //处理用户登录
    public String execute() throws Exception{
        UserDao userDao = new UserDaoImpl();
        User user = userDao.select(getModel().getUsername());
        if(null != user ){

    if(getModel().getPassword().equalsIgnoreCase(user.getPassword())){
                getModel().setTip("服务器提示！！！");
                //String user = (String)
ActionContext.getContext().getSession().get("user");
                ActionContext.getContext().getSession().put("user",
getModel().getUsername());
                return "success";
            }else{
                getModel().setTip("输入用户名或密码错误！！！");
                return "error";
            }
            }else{
                getModel().setTip("没有该用户！！！");
                return "error";
            }
        }
    }
```

(6)　打开项目 src 目录下的 struts.xml 文件，在该文件中添加 UserAction 的配置信息，代码如下所示。

```
<struts>
    <constant name="struts.i18n.encoding" value="UTF-8"/>
    <constant name="struts.custom.i18n.resources" value="messageResource"/>
    <package name="mystrut2" extends="struts-default">
        <action name="Login" class="action.UserAction">
            <result name="input">/login.jsp</result>
            <result name="error">/error.jsp</result>
            <result name="success">/login_success.jsp</result>
        </action>
        <action name="register" class="action.UserAction" method="register">
            <result name="error">/error.jsp</result>
            <result name="success">/login.jsp</result>
        </action>
    </package>

</struts>
```

(7)　新建一个 register.jsp 页面，在该页面中定义一个 form 表单，供用户输入注册信息，单击“注册”按钮，发出注册请求，详细实现代码如下所示。

```
<body topmargin="30%">
    <s:form action="register" method="post">
    <table border="1" cellspacing="0" align="center">
        <caption align="center">用户注册</caption>
        <tr>
            <td><s:textfield name="username" label="用户名"/></td>
        </tr>
        <tr>
            <td><s:textfield name="password" label="密码"/></td>
        </tr>
        <tr>
            <td><s:textfield name="truename" label="真实姓名"/></td>
        </tr>
        <tr>
            <td><s:textfield name="sex" label="性别"/></td>
        </tr>
        <tr>
            <td><s:textfield name="job" label="职位"/></td>
        </tr>
        <tr>
            <td><s:submit value="注册"/></td><td><s:reset value="重置"/></td>
        </tr>
    </table>
    </s:form>
</body>
```

(8) 新建一个 login.jsp，在该页面定义一个用于用户登录的表单，当用户输入用户名密码，单击“登录”按钮时，发出登录请求，代码如下所示。

```
<body>
    <s:form action="Login.action">
        <h3>用户登录</h3>
        <s:textfield name="username" label="用户名: "/>
        <s:textfield name="password" label="密码: "/>
        <s:submit value="提交"/>
    </s:form>
</body>
```

(9) 新建一个 login_success.jsp 页面，当用户登录成功时，进入登录成功页面，并显示“欢迎，***，您已经登录成功！”，代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@taglib prefix="s" uri="/struts-tags"%>
<html>
    <head>
        <title>登录页面</title>
    </head>
    <body>
        <h3><s:property value="tip"/></h3>
        欢迎，${sessionScope.user}，您已经登录成功!
    </body>
</html>
```

(10) 新建一个 error.jsp 页面，当用户登录失败时，给出错误提示。代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@taglib prefix="s" uri="/struts-tags"%>
<html>
    <head>
        <title>错误</title>
    </head>
    <body>
        <s:property value="tip"/>
    </body>
</html>
```

11.1.4 运行结果

打开 IE 浏览器，在地址栏中输入“http://localhost:8080/Struts2_11/register.jsp”，进入注册页面，执行效果如图 11-4 所示。

当用户输入注册信息完毕，单击“注册”按钮，将进入登录界面可以进行登陆，执行效果如图 11-5 所示。

单击“登录”按钮，登录成功后，进入系统欢迎用户页面，如图 11-6 所示。



图 11-4　用户注册页面

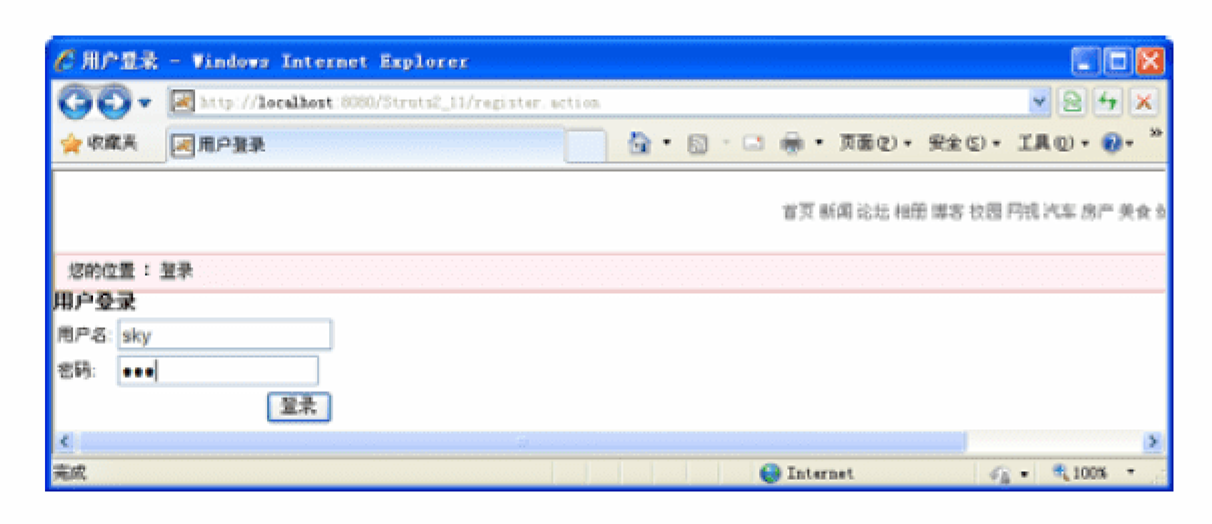
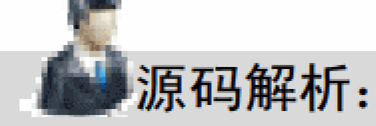


图 11-5　用户登录页面



图 11-6　欢迎用户登录

11.1.5　实例分析

**源码解析：**

本实例中在持久层定义了一个 User 实体类，并在 dao 层定义了一个操作 User 对象的接口 UserDao。在该接口中声明了一个查询用户对象的 select()方法和添加用户对象的 save()方法，并在 UserDaoImpl 类中实现了该接口。

创建一个 UserAction 用于处理用户的注册和登录请求。当用户注册成功时，跳转到登录页面 login.jsp 进行登录，登录成功后将进入欢迎用户界面，可以享受用户相关的其他服务。

11.2　添 加 用 户

Spring 在英文里有春天、弹簧、跳跃和泉眼的意思。Spring 也表示一个开源框架，是为了

解决企业应用程序开发复杂性，由 Rod Johnson 创建的。框架的主要优势之一是其分层架构，分层架构允许使用某一个组件，同时为 J2EE 应用程序开发提供集成的框架。

视频教学：光盘/videos/11/struts_spring.avi　　长度：6 分钟
光盘/videos/11/struts_spring1.avi　　长度：12 分钟

11.2.1　基础知识——集成Spring

前面已经提到过，Struts 2 通过插件的方式对 Spring 提供支持。Spring 的依赖注入功能以及对 ORM 和 DAO 的支持，可以用来简化 Hibernate 的配置和访问操作。本节将介绍这三个框架的集成开发。

1. Spring概述

Spring 使用基本的 JavaBean 来完成以前只可能由 EJB 完成的事情。Spring 的用途不仅限于服务器端的开发。从简单性、可测试性和松耦合的角度而言，任何 Java 应用都可以从 Spring 中受益。

传统 J2EE 应用的开发效率低，应用服务器厂商对各种技术的支持并没有真正统一，导致 J2EE 的应用没有真正实现 Write Once 及 Run Anywhere 的承诺。Spring 作为开源的中间件，独立于各种应用服务器，甚至无须应用服务器的支持，也能提供应用服务器的功能，如声明式事务等。

Spring 致力于 J2EE 应用的各层的解决方案，而不是仅仅专注于某一层的方案。可以说 Spring 是企业应用开发的“一站式”选择，并贯穿表现层、业务层及持久层。然而，Spring 并不想取代那些已有的框架，而是与它们无缝地整合。

2. Spring带来的好处与优点

在进入细节以前，先看一下 Spring 可以给一个项目带来的一些好处。

- Spring 能有效地组织中间层对象，无论你是否选择使用了 EJB。如果你仅仅使用了 Struts 或其他的包含了 J2EE 特有 APIs 的 Framework，你会发现 Spring 关注了遗留下的问题。
- Spring 能消除在许多项目上对 Singleton 的过多使用。这是一个主要的问题，它减少了系统的可测试性和面向对象特性。
- Spring 能消除使用各种各样格式的属性定制文件的需要，在整个应用和项目中，可通过一种一致的方法来进行配置。曾经感到迷惑，一个特定类要查找迷幻般的属性关键字或系统属性，为此不得不读 Javadoc 乃至源编码吗？有了 Spring，你可以很简单地看到类的 JavaBean 属性。倒置控制的使用可以(在下面讨论)帮助完成这种简化。
- Spring 能通过接口而不是类促进好的编程习惯，将编程代价减少到几乎为零。
- Spring 被设计为让使用它创建的应用尽可能少的依赖于它的 APIs。在 Spring 应用中，大多数业务对象没有依赖于 Spring。

- 使用 Spring 构建的应用程序易于单元测试。
- Spring 能使 EJB 的使用成为一个实现选择，而不是应用架构的必然选择。可以选择用 POJOs 或 local EJBs 来实现业务接口，从而不会影响调用代码。
- Spring 帮助你无需使用 EJB 就能解决许多问题。Spring 能提供一种 EJB 的替换物，它们适于许多 web 应用。例如：Spring 能使用 AOP 提供声明性事务而不通过使用 EJB 容器，如果仅仅需要与单个的数据库打交道，甚至不需要 JTA 实现。
- Spring 为数据存取提供了一致的框架，不论是使用 JDBC 还是使用 O/R mapping 产品(如 Hibernate)。

Spring 使你通过最简单可行的办法解决你的问题。这些特性具有很大价值。

总结起来，Spring 有如下优点。

- 低侵入式设计，代码污染极低。
- 独立于各种应用服务器，可以真正实现“一次编写，处处运行”的承诺。
- Spring 的 DI 机制降低业务对象替换的复杂性。
- Spring 并不强制 Web 应用完全依赖于 Spring，开发者可自由选用 Spring 框架的部分或全部。
- Spring 为数据存储提供了一致的框架，不论是使用 JDBC，还是使用 ORM(例如 Hibernate)。

3. 下载安装Spring

在 Web 应用中使用 Spring 框架，需要先在应用中放置 Spring 的相关 JAR 文件。下载和安装 Spring，可以按如下步骤进行。

(1) 登录 Spring 官方网站“http://www.springsource.org”站点，下载 Spring 的最新稳定版本。本书中采用的版本是 spring-framework-2.5.5-with-dependencies.zip。

(2) 下载完成后，解压 zip 包，解压缩后的 spring-framework-2.5.5 文件夹中有如下几个文件夹。

- dist：该文件夹下放 Spring 的 jar 包，通常只需要 Spring.jar 文件即可。该文件夹下还有一些类似 spring-Xxx.jar 的压缩包，这些压缩包是 spring.jar 压缩包的子模块压缩包。除非确定整个 J2EE 应用只需要使用 Spring 的某一方面时，才考虑使用这种分模块压缩包。通常建议使用 Spring.jar。
- docs：该文件夹下包含 spring 的相关文档、开发指南及 API 参考文档。
- lib：该文件夹下包含 spring 编译和运行所依赖的第三方类库，该路径下的类库并不是 spring 必需的，但如果需要使用第三方类库的支持，这里的类库就是需要的。
- samples：该文件夹下包含 Spring 的几个简单例子，可作为 Spring 入门学习的案例。
- src：该文件夹下包含 Spring 的全部源文件，如果开发过程中有些地方无法把握，可以参考该源文件，了解底层实现。
- test：该文件夹下包含 Spring 的测试示例。
- tiger：该路径下存放关于 JDK 的相关内容。

● 其他相关文件：解压缩后的文件夹下，还包含一些关于 Spring 的 License 和项目相关文件。

(3) 将 spring.jar 复制到项目的 CLASSPATH 路径下，对于 Web 应用，将 spring.jar 文件复制到 WEB-INF/lib 路径下，该应用即可以利用 Spring 框架了。

(4) 通常 Spring 的框架还依赖于其他一些 jar 文件，因此还须将 lib 下对应的包复制到 WEB-INF/lib 路径下，具体要复制哪些 jar 文件，取决于应用所需要使用的项目。通常需要复制 cglib，dom4j，jakarta-commons，log4j 等文件夹下的 jar 文件。

4．Spring框架组件

Spring 框架是一个分层架构，由 7 个定义良好的组件组成。Spring 组件构建在核心容器之上，核心容器定义了创建、配置和管理 Bean 的方式。组成 Spring 框架的每个组件都可以单独存在，或者与其他一个或多个组件联合实现。这 7 个组件如下。

1) Spring Core

Spring Core 是 Spring 的核心容器，它提供 Spring 框架的基本功能。核心容器的主要组件是 BeanFactory，它是工厂模式的实现。BeanFactory 使用 IoC 模式将应用程序的配置和依赖性规范与实际的应用程序代码分开。

2) Spring Context

Spring Context 是一个配置文件，向 Spring 框架提供上下文信息。



Spring Context 包括企业服务，例如：JNDI、EJB、电子邮件、国际化、校验和调度功能等。

3) Spring AOP

通过配置管理特性，Spring AOP 直接将面向切面的编程功能集成到了 Spring 框架中。所以，可以很容易使 Spring 框架管理的任何对象支持 AOP。



Spring AOP 为基于 Spring 的应用程序中的对象提供了事务管理服务。通过使用 Spring AOP，不用依赖 EJB 组件，就可以将声明性事务管理集成到应用程序中。

4) Spring DAO

JDBC DAO 抽象层提供了有意义的异常层次结构，可以用该结构来管理异常处理和不同数据库供应商抛出的错误消息。Spring DAO 的面向 JDBC 的异常遵从通用的 DAO 异常层次结构。



异常层次结构简化了错误处理，并且极大地降低了需要编写的异常代码数量(例如打开和关闭连接)。

5) Spring ORM

Spring 框架插入了若干个 ORM 框架，从而提供了 ORM 的对象关系工具。

6) Spring Web

Spring 框架支持与 Jakarta Struts 的集成。Spring Web 组件简化了处理多部分请求以及将请求参数绑定到域对象的工作。

7)　Spring Web MVC

Spring MVC 框架是一个全功能的构建 Web 应用程序的 MVC 实现。通过策略接口，Spring MVC 框架变成为高度可配置的。



Spring MVC 容纳了大量视图技术，其中包括 JSP、Velocity 和 Tiles 等。

Spring 框架的组件结构如图 11-7 所示。

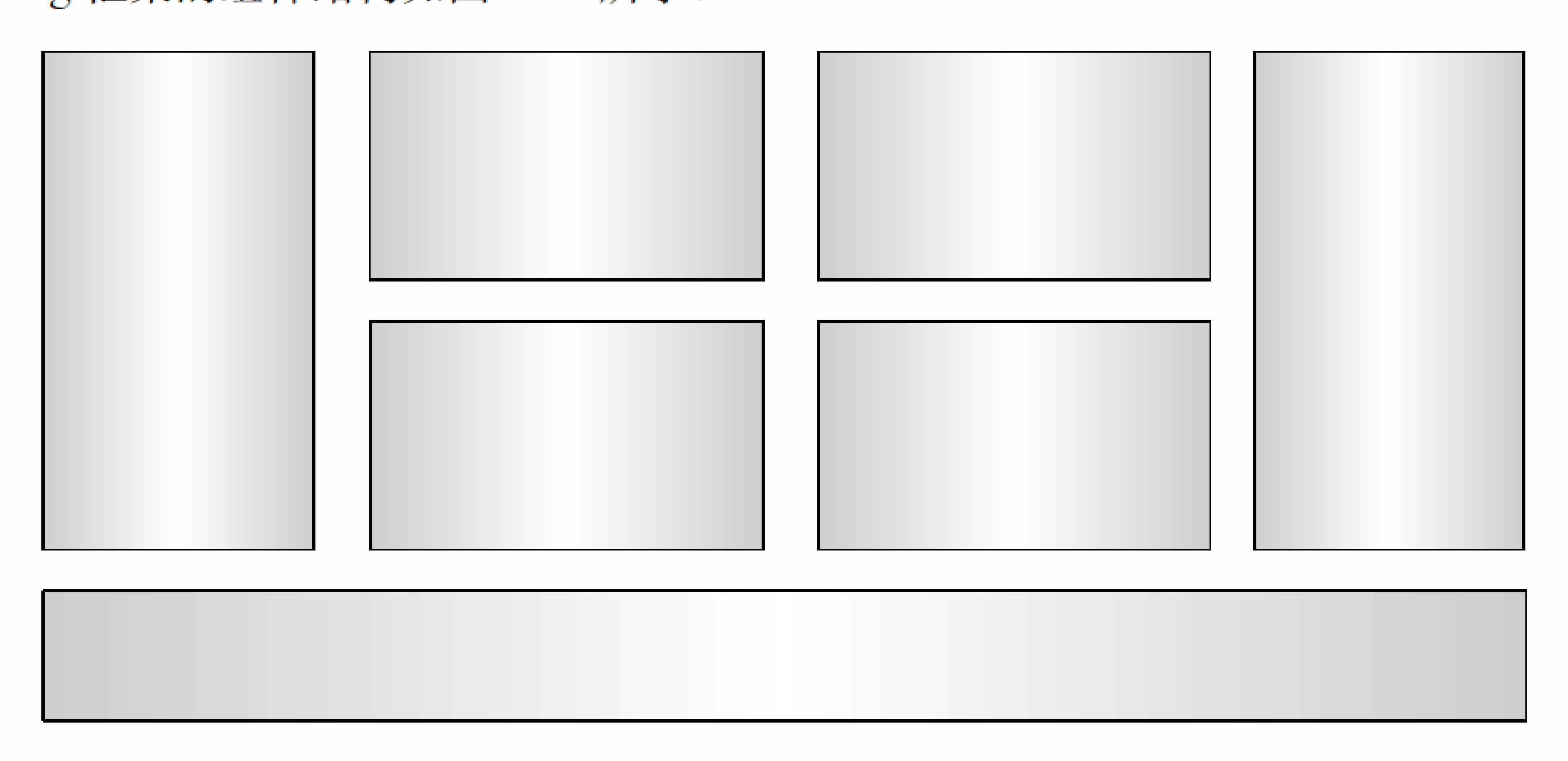


图 11-7　Spring框架的组件结构

5．Spring的IoC容器

IoC，全称(Inverse Of Control)，中文意思为：控制反转，Spring 框架的核心基于控制反转原理。

什么是控制反转？控制反转是一种将组件依赖关系的创建和管理置于程序外部的技术。由容器控制程序之间的关系，而不是由代码直接控制。由于控制权由代码转向了容器，所以称为反转。

在 Spring 中，当一个角色 A 需要另一个角色 B 协助工作时，Spring 容器负责调用 B，而不需要 A 自己去调用，这被称为控制反转；Spring 容器创建 B 的实例，然后注入给 A，称为依赖注入。这两者说的是同一件事情。

(1)　下面举一个简单的例子来介绍依赖注入的使用。创建一个 Person 接口，定义一个 useAxe()方法，代码如下所示。

```
package com.IoC;

public interface Person {
    /**
     * 使用斧子
     */
    public void useAxe();
}
```

Spring AOP
Source-level
metadata

(2) 创建一个 Axe 接口，定义一个 chop()方法，代码如下所示。

```
package com.IoC;
public interface Axe {
    /**
     * 砍的方法
     */
    public String chop();
}
```

(3) 创建一个 Chinese 类实现 Person 接口，代码如下所示。

```
package com.IoC;

public class Chinese implements Person{
    private Axe axe;
    /**
     * 设值注入所需的 setter 方法
     * @param axe
     */
    public void setAxe(Axe axe)
    {
        this.axe = axe;
    }
    public void useAxe() {
        System.out.println(axe.chop());
    }
}
```

(4) 创建一个 StoneAxe 类实现 Axe 接口，代码如下所示。

```
package com.IoC;
public class StoneAxe implements Axe{
    public String chop() {
        return "这是一把砍得很慢的破斧子！";
    }

}
```

(5) 在 Spring 的配置文件中对实现类进行配置，配置文件的内容如下代码所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
<!-- 以上是全部 Spring 都一样的配置文件 、beans 是根元素-->
    <bean name="chinese" class="com.IoC.Chinese">
        <property name="axe">
<!-- 如果注入的是基本类型则用 value、如果是引用类型则使用 ref -->
            <ref local="StoneAxe"/>
        </property>
```

```
    </bean>
    <bean id="StoneAxe"  class="com.IoC.StoneAxe"></bean>
</beans>
```



StoneAxe 类有一个使用斧子的方法，而 Chinese 类是使用斧子的人。XML 配置文件里面声明两个 bean，但是 StoneAxe 是 Chinese 的一个属性，但是 value 用于基本类型，而 ref 用于引用类型。运行 Chinese 的时候，提供调用 useAxe()进行使用“斧子”，要引进 StoneAxe 则通过 seter 方法，进行后面的属性注入。

11.2.2　实例描述

刚登上 QQ，就是铺天盖地的群信息。看看这都是谁发的，一看又是老一套，信息上是在求助帮忙找一个 Struts 2、Hibernate 和 Spring 集成的项目源码，说是紧急使用。看他着实可怜，心中实在不忍，于是我给他写了一个添加用户的例子，现在拿出来和读者共同学习一下。实际上这三大框架的整合并没有读者想象的那么难，看看下面的实例就知道了。

11.2.3　实例应用

【例 11-2】 添加用户。

(1) 创建一个 Struts2_SH 项目，然后将 Struts 2、Hibernate、Spring 所需 JAR 包复制粘贴到项目的 WEB-INF/lib 目录下。最后在项目的 src 目录下新建一个 com.test.bean 包，在该包下新建一个用户实体类 User，包含 firstname、lastname 和 age 属性，代码如下所示。

```
public class User {
    private int id;               //用户编号
    private String firstname;    //用户的姓
    private String lastname;     //用户的名
    private int age;              //用户的年龄
}
```

(2) 在 User 实体类所在目录下，新建一个 User.hbm.xml 文件来映射 User 类，代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.test.bean.User" table="users">
        <id name="id">
        <generator class="native" />
        </id>
        <property name="firstname" ></property>
        <property name="lastname"></property>
```

```
        <property name="age"></property>
    </class>
</hibernate-mapping>
```

(3) 设计项目的 DAO 层，在项目的 src 目录下新建一个 com.test.dao 包，在该包下新建一个 UserDAO 接口，声明一个添加用户方法和一个查询所有用户信息的方法，代码如下所示。

```
package com.test.dao;

import java.util.List;
import com.test.bean.User;

public interface UserDAO {
    //添加一个用户
    public void saveUser(User user);
    //查询所有用户信息
    public List<User> findAllUsers();
}
```

(4) 在项目 src 目录下，新建一个 com.test.dao.impl 包，在该包下新建一个 UserDAOImpl 类，该类需要实现 UserDAO 接口，代码如下所示。

```
public class UserDAOImpl extends HibernateDaoSupport implements UserDAO {
    //查询所有用户信息
    @SuppressWarnings("unchecked")
    public List<User> findAllUsers() {
        String hql="from User";
        return (List<User>)this.getHibernateTemplate().find(hql);
    }
    //添加一个用户
    public void saveUser(User user) {
        this.getHibernateTemplate().save(user);
    }
}
```

(5) 在项目 src 目录下，新建一个 com.test.service 包，在该包下定义一个业务逻辑层接口 UserService，处理添加用户和查询所有用户信息的业务逻辑，代码如下所示。

```
package com.test.service;
import java.util.List;
import com.test.bean.User;

public interface UserService {
    //查询所有用户信息
    public List<User> findAll();
    //添加一个用户
    public void save(User user);
}
```

(6)　在项目 src 目录下，新建一个 com.test.service.impl 包，在该包下定义一个 UserService Impl 类，该类需要实现业务逻辑接口 UserService，代码如下所示。

```
package com.test.service.impl;

import java.util.List;
import com.test.bean.User;
import com.test.dao.UserDAO;
import com.test.service.UserService;

public class UserServiceImpl implements UserService {

    private UserDAO userDao;    //声明一个 UserDao 对象

    public UserDAO getUserDao() {
        return userDao;
    }

    public void setUserDao(UserDAO userDao) {
        this.userDao = userDao;
    }
    //查询所有用户信息
    public List<User> findAll() {
        return this.userDao.findAllUsers();
    }
    //保存一个用户
    public void save(User user) {
        this.userDao.saveUser(user);
    }

}
```

(7)　业务逻辑层完成之后，开始控制器层的设计。在项目的 src 目录下，新建一个 com.test.action 包，在该包下新建一个 SaveUserAction，用于处理添加保存用户请求，代码如下所示。

```
package com.test.action;

import javax.servlet.http.HttpServletRequest;
import org.apache.struts2.ServletActionContext;
import com.opensymphony.xwork2.ActionSupport;
import com.test.bean.User;
import com.test.service.UserService;

public class SaveUserAction extends ActionSupport {

    private User user = new User();//创建一个用户对象
    private UserService service;       //创建处理用户操作业务对象
    public User getUser() {
```

```
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public UserService getService() {
        return service;
    }

    public void setService(UserService service) {
        this.service = service;
    }

    @Override
    public String execute() throws Exception {
        this.service.save(this.user);      //保存用户
        return SUCCESS;
    }
}
```

(8) 在 com.test.action 包下，再新建一个 UserListAction，用于查询获取所有用户信息，代码如下所示。

```
package com.test.action;

import java.util.List;
import com.opensymphony.xwork2.ActionSupport;
import com.test.bean.User;
import com.test.service.UserService;

public class UserListAction extends ActionSupport{
    private UserService service;        //创建处理用户操作业务对象
    private List<User> list;        //创建一个存储用户信息的列表
    public List<User> getList() {
        return list;
    }
    public void setList(List<User> list) {
        this.list = list;
    }
    public UserService getService() {
        return service;
    }
    public void setService(UserService service) {
        this.service = service;
    }
    @Override
    public String execute() throws Exception {
```

```
        list = this.service.findAll();      //查询获取所有用户信息列表
        return "success";
    }
}
```

(9) 在项目 src 目录下新建一个 struts.xml 文件。在该文件中添加 SaveUserAction 和 UserListAction 的配置，代码如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="struts2" extends="struts-default">
        <action name="SaveUser" class="saveuseraction">
            <result name="success" type="redirect">list.action</result>
            <result name="input">/save.jsp</result>
        </action>
        <action name="list" class="userlistaction">
            <result>/list_user.jsp</result>
        </action>
    </package>
</struts>
```

(10) 上述是对 Action 的配置，下面使用 Spring 来配置 Hibernate 和 UserDAO、UserDAOImpl、UserService 和 UserServiceImpl 一系列 bean，还有 Action 的注入配置，代码如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
<!-- 连接数据库的数据源的配置 -->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName"
value="com.mysql.jdbc.Driver"></property>
    <property name="url"
value="jdbc:mysql://localhost:3306/spring"></property>
    <property name="username" value="root"></property>
    <property name="password" value="123456"></property>
    <property name="maxActive" value="100"></property>
    <property name="maxIdle" value="30"></property>
    <property name="maxWait" value="500"></property>
    <property name="defaultAutoCommit" value="true"></property>
</bean>
<!-- Hibernate 的 sessionFactory 的配置 -->
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
```

```
    <property name="dataSource" ref="dataSource"></property>
    <property name="hibernateProperties">
        <props>
            <prop
key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
            <prop key="hibernate.show-sql">true</prop>
        </props>
    </property>
    <property name="mappingResources">
        <list>
            <value>com/test/bean/User.hbm.xml</value>
        </list>
    </property>
</bean>
<!-- UserDAOImpl 的注入配置 -->
<bean id="userDao" class="com.test.dao.impl.UserDAOImpl">
    <property name="sessionFactory">
        <ref bean="sessionFactory"/>
    </property>
</bean>
<!-- UserServiceImpl 的注入配置 -->
<bean id="userService" class="com.test.service.impl.UserServiceImpl">
    <property name="userDao">
        <ref bean="userDao"/>
    </property>
</bean>
<!--SaveUserAction 的注入配置 -->
<bean id="saveuseraction" class="com.test.action.SaveUserAction">
    <property name="service" ref="userService"></property>
</bean>
<!-- UserListAction 的注入配置 -->
<bean id="userlistaction" class="com.test.action.UserListAction">
    <property name="service" ref="userService"></property>
</bean>
</beans>
```

(11) 打开项目 WEB-INF 目录下的 web.xml 文件，在该文件中添加 spring 的 Context LoaderListener 监听器和 Struts 2 的 FilterDispatcher 拦截器，代码如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <listener>

    <listener-class>org.springframework.web.context.ContextLoaderListener</
listener-class>
```

```
    </listener>
    <filter>
        <filter-name>struts2</filter-name>

    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-cl
ass>
    </filter>
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

(12) 上述配置完毕，在项目目录下新建一个 index.jsp 页面，该页面可以为用户提供一个添加用户链接和一个查询所有用户信息链接，代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>用户系统</title>
  </head>
  <body>
    <s:a href="save.jsp">添加用户</s:a><br><br/>
    <s:a href="list.action">查询所有用户信息</s:a>
  </body>
</html>
```

(13) 当用户单击“添加用户”链接时，跳转到 save.jsp 页面，该页面包含一个用户信息表单，输入用户信息之后，可单击“保存”按钮，保存用户信息，代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>添加用户</title>
    </head>
    <body>
        <s:form action="SaveUser">
            <s:textfield name="user.firstname" label="姓"></s:textfield>
            <s:textfield name="user.lastname" label="名"></s:textfield>
            <s:textfield name="user.age" label="年龄"></s:textfield>
            <s:submit value="保存"></s:submit>
        </s:form>
    </body>
</html>
```

(14) 当保存用户成功时，将转发到 list.action 查询显示用户信息，查询成功页面跳转到

list_user.jsp 页面，显示所有用户的信息，代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>显示用户信息</title>
    </head>
    <body>
        <table>
            <s:iterator value="list" id="name" status="status">
                <tr <s:if
test="#status.odd">style="background-color:yellow"</s:if>>
                    <td><s:property value="#status.count"/></td>
                    <td>姓：<s:property value="firstname"/></td>
                    <td>名：<s:property value="lastname"/></td>
                    <td>年龄：<s:property value="age"/></td>
                </tr>
            </s:iterator>
        </table>
    </body>
</html>
```

11.2.4 运行结果

打开 IE 浏览器，在地址栏中输入“http://localhost:8080/Struts2_SH/index.jsp”，进入用户管理系统，可以添加用户和查看用户信息，执行效果如图 11-8 所示。

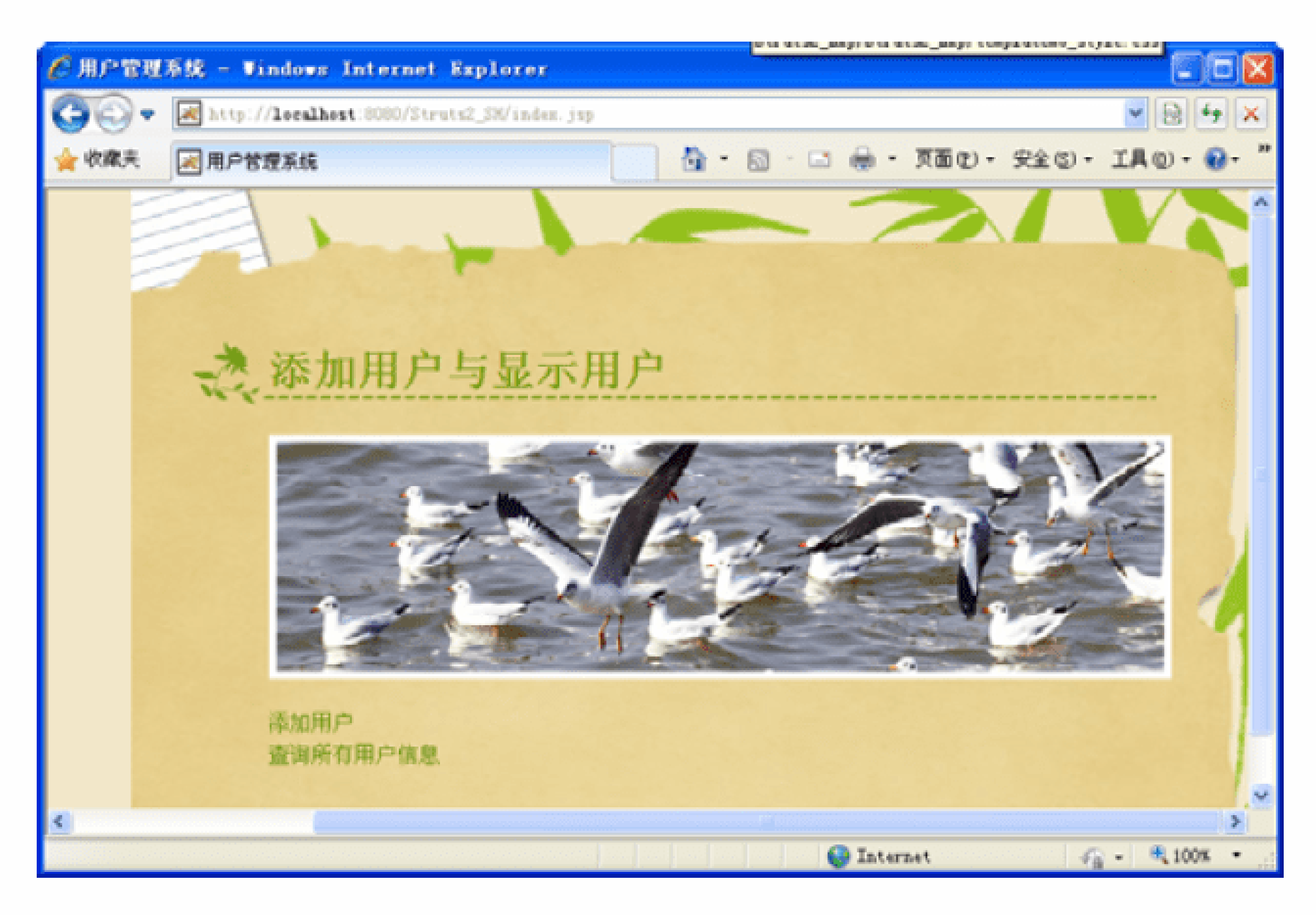


图 11-8 用户管理系统

单击“添加用户”链接，可以添加一个新用户，单击“查询所有用户信息”链接，可以查看用户信息，添加用户执行效果如图 11-9 所示。

添加用户完成之后，页面跳转到显示用户信息页面，执行效果如图 11-10 所示。



图 11-9　添加用户

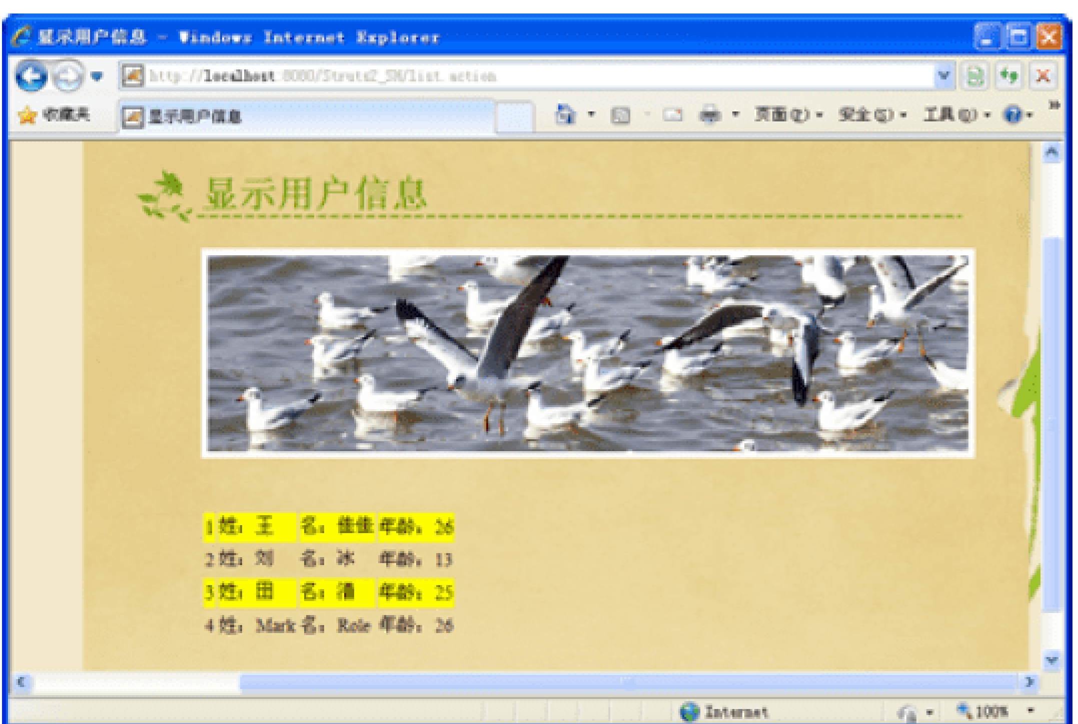


图 11-10　显示用户信息

11.2.5　实例分析



源码解析：

本实例主要是操作用户，首先创建一个 User 实体类，并编写 User 类的映射数据库表的配置文件 User.hbm.xml。设计 DAO 层创建 UserDAO 接口，声明一个添加用户方法和一个查询所有用户信息方法，并新建 UserDAOImpl 类实现 UserDAO 接口，业务逻辑层新建一个 UserService 接口，声明两个业务逻辑方法，添加用户方法和查询所有用户信息方法，再新建一个 UserServiceImpl 类实现 UserService 接口。

然后实现控制器层，编写两个 Action：SaveUserAction 和 UserListAction，分别用于处理添加用户请求和查询用户请求。并在 struts.xml 文件中添加这两个 Action 的相应配置信息。

最后添加 applicationContext.xml 配置文件，配置 Hibernate 相关信息和 DAO 层、业务逻辑层和控制器层类的注入配置。再新建一个 save.jsp 页面用于保存用户，list_user.jsp 页面用于显示用户信息。

11.3　常见问题解答

11.3.1　Struts 2+Hibernate+Spring整合错误严重：Exception starting filter struts 2



Struts 2+Hibernate+Spring 整合错误严重:Exception starting filter struts 2?
网络课堂：http://bbs.itzcn.com/thread-11007-1-1.html

Struts 2+Hibernate+Spring 整合项目，运行时，控制台输出如下错误信息。

```
严重: Exception starting filter struts2
java.lang.RuntimeException: java.lang.RuntimeException:
java.lang.RuntimeException:
com.opensymphony.xwork2.inject.DependencyException:
com.opensymphony.xwork2.inject.ContainerImpl$MissingDependencyException: No
mapping found for dependency [type=java.lang.String,
name='struts.objectFactory.spring.autoWire.alwaysRespect'] in public
org.apache.struts2.spring.StrutsSpringObjectFactory(java.lang.String,java.l
ang.String,java.lang.String,javax.servlet.ServletContext).
```

不知该如何解决，请各位高手帮忙解答。

【解决办法】：出现这种错误，可能是因为如下几种原因。

(1) 有可能是 jar 包互相冲突，这个没有好办法，只有一个一个去试了。

(2) 有可能是 Struts 2 的配置文件里面的 Action 的路径写错了，查看一下 struts.xml 文件。

(3) 也有可能是 web.xml 中的 filter 写错了。web.xml 里加入如下代码。

```
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</list
ener-class>
</listener>
```

还必须添加 struts2-spring-plugin-2.1.6.jar 包。

11.3.2 出现java.lang.NoClassDefFoundError问题



出现 java.lang.NoClassDefFoundError:org 问题？
网络课堂：http://bbs.itzcn.com/thread-11017-1-1.html

我第一次整合 Spring+Hibernate+Struts 2 项目，整合完毕之后运行，在控制台输出“java.lang.NoClassDefFoundError:org/slf4j/LoggerFactory”错误，不知该如何解决。

【解决办法】：出现这个错误是因为项目中缺少 jar 包，你可以加入 slf4j-api-1.5.0.jar、slf4j-log4j12-1.5.0.jar、log4j-1.2.15.jar 包试一试。

11.3.3 org.hibernate.id.IdentifierGenerationException异常问题



org.hibernate.id.IdentifierGenerationException 是什么异常，如何解决？
网络课堂：http://bbs.itzcn.com/thread-11022-1-1.html

我从网上下载了一个 Struts 2、Spring 和 Hibernate 整合项目，搭建了开发环境，运行时，在控制台输出“org.hibernate.id.IdentifierGenerationException”异常，不知道是什么意思，该如何解决？紧急求解！

【解决办法】：出现这个异常的原因是：你的实体类映射文件中的<id>元素配置不正确，

<id>元素缺少其子元素<generator></generator>的配置。

<id>元素映射了相应数据库表的主键字段，对其子元素<generator class="">中的 class 取值可以为 increment、identity、sequence、hilo、native 等，更多的可参考 hibernate 参考文档，一般取其值为 native 功能是适应本地数据库。

11.4 习　　题

一、填空题

(1) 在 AOP 中，通知分为前置通知、后置通知、____________通知和异常通知。

(2) Hibernate 支持两种主要的查询方式: ____________查询和 Criteria 查询。

(3) 在 Hibernate 配置文件中，<set>节点的____________属性描述了级联操作的规则。

(4) 在 SSH 中，使用____________框架实现数据持久化。

(5) 在 SSH 中，使用____________框架管理依赖关系。

二、选择题

(1) 关于 Spring 对 Hibernate 的支持，下面说法错误的是____________。(单选)

A. Spring 提供基类完成了繁琐的异常处理代码

B. Spring 提供基类完成了繁琐的事物处理代码

C. Spring 提供的基类对查询提供良好的支持

D. Spring 提供的基类需要注入 sessionFactory 才能正常运行

(2) 下面关于 AOP 的说法错误的是____________。(单选)

A. AOP 将散落在系统中的“方面”代码集中实现

B. AOP 有助于提高系统的可维护性

C. AOP 已经表现出了将要代替面向对象的趋势

D. AOP 是一种设计模式，Spring 提供了一种实现

(3) 对 Struts 的描述，错误的是____________。(单选)

A. Struts 基于 Servlet 技术实现

B. 使用 Struts 时不能同时使用 Hibernate 或 Spring，也不能在页面使用 EL 表达式

C. Struts 是 MVC 设计模式的实现

D. Struts 是一个半成品，可以基于它构建自己的应用程序

(4) 某 blog 系统采用三层结构组织程序，访问数据库的代码和定义积分规则的代码分别应该放在系统的____________。(单选)

A. 数据访问层和表现层　　　　B. 持久化层和业务逻辑层

C. 数据访问层和业务逻辑层　　D. 持久化层和验证层

三、上机练习

上机练习：添加并显示员工信息列表。

要求：该项目首先创建一个 Employee 实体类，编写该类的映射配置信息，设计 DAO 层的

EmployeeDAO 接口，主要定义添加员工方法和查询员工信息方法，并实现该接口。接下来设计业务逻辑层的 EmployeeService 接口添加员工方法和查询员工信息方法，并实现该接口。控制器层定义一个 EmployeeAction 包含添加员工和查询员工信息操作。还需要设计视图层 add.jsp 用于添加员工，list.jsp 用于显示员工信息。

执行效果如图 11-11 和图 11-12 所示。



图 11-11　添加员工



图 11-12　显示员工信息



第 12 章　整合JFreeChart

内容摘要：

对于一个企业级的应用而言，常常需要生成大量的统计图表，例如，饼图、柱状图等。生成这些统计图通常有两种做法：一种是直接使用 Applet 作为容器来显示这些统计图，一种是临时生成统计图的图片，并在 HTML 页面中显示出这些图片。这两种的实现方式都存在着效率低下、需要大量处理底层的图形细节等问题。

有了 JFreeChart 的包装，就可以无须开发者自己来处理底层的图形细节。借助于 JFreeChart 的帮助，开发者可以非常便捷地开发出各种各样的图表。这些图表包括：饼图、柱状图(普通柱状图及堆栈柱状图)、线图、区域图、分布图、混合图、甘特图及一些仪表盘等。大部分企业级应用所需要的统计图，JFreeChart 基本都可以满足。

这一章将为读者介绍使用 JFreeChart 如何生成饼图、柱状图、折线图、时间顺序图及带交互功能的热点统计图，并详细介绍 Struts 2 与 JFreeChart 整合。

学习目标：

- 熟练使用 JFreeChart 生成饼图。
- 熟练使用 JFreeChart 生成饼状图。
- 熟练使用 JFreeChart 生成折线图。
- 熟练使用 JFreeChart 生成时间顺序图。
- 熟练使用 JFreeChart 生成带交互功能的热点统计图。
- 掌握 Struts 2 与 JFreeChart 整合。

12.1 初始JFreeChart

JFreeChart 是完全基于 Java 语言的开源项目，因此可以在 Java 开发环境中使用 JFreeChart，包括 Java 应用程序，或者是 Java Web 应用都没有任何问题。结合 iText 项目，也可以将生成的统计图表输出到 PDF 文件；结合最新的 POI 项目，也可以将生成的统计图表图输出到 Excel 文档。这一节将为读者介绍如何使用 JFreeChart 开发项目。

视频教学：光盘/videos/12/JFreeChart.avi　　长度：5 分钟

12.1.1 基础知识——初始JFreeChart

JFreeChart 可用于生成各种各样的统计图表，只要开发者提供符合 JFreeChart 所需格式的数据，JFreeChart 即可自动生成相应的统计图表，这些统计图表既可以直接输出生成图片文件，也可被导出成 PDF 文档。

1. JFreeChart的下载和安装

使用 JFreeChart 来生成统计图表，必须下载和安装 JFreeChart 项目。下载和安装 JFreeChart 的步骤如下。

(1) 登录 JFreeChart 的官方下载站点：http://www.jfree.org/jfreechart/download.html，下载 JFreeChart 的最新版本。本书使用的版本是 JFreeChart 1.0.13，JFreeChart 依赖于另外一个项目：JCommon；同时下载该项目的最新版本：JCommon 1.0.16。

下载 JFreeChart 有如下三个选项。

- JFreeChart：下载 JFreeChart 项目的压缩文件。
- Documentation：下载 JFreeChart 的 API 文档压缩文件。
- JCommon：下载 JCommon 项目的压缩文件。

依次下载三个选项，这三个选项都是使用 JFreeChart 项目所需要的。



JCommon 选项可以不用下载，但如果开发者需要查阅 JCommon 的相关资料，例如查看 JCommon 项目的源文件，则应该下载该选项。

(2) 下载上面的三个选项后，会得到三个压缩文件：jfreechart-1.0.13.zip 文件、jfreechart-1.0.13-javadocs.zip 文件和 jcommon-1.0.16.zip 文件。其中 jfreechart-1.0.13-javadocs.zip 文件里包含了 JFreeChart 项目的 API 文档，与其他项目的 API 文档完全相似，此处不再赘述。

解压缩 jfreechart-1.0.13.zip 文件，得到如图 12-1 所示的文件结构。

其中：

- ant：该文件夹下存放了编译 JFreeChart 项目的 build.xml 文件。
- checkstyle：该文件夹下存放了生成 JFreeChart 项目 API 文档的样式文件。
- docfiles：该文件夹下存放了生成 JFreeChart 项目的图表、图片。
- experimental：该文件夹下存放的是 JFreeChart 项目的实验性新功能的源代码。

图 12-1　JFreeChart项目文档结构

- lib：该文件夹下存放的是 JFreeChart 项目的二进制类库，以及编译和运行 JFreeChart 所依赖的第三方类库。
- source：该文件夹下存放的是 JFreeChart 的源代码。
- swt：该路径下存放的是 JFreeChart 提供的 SWT(Java Standard　Widger Toolkit，Java 标准工具集)支持的源代码。
- tests：该路径下存放的是 JFreeChart 项目单元测试的测试用例文件。
- jfreechart-1.0.13-demo.jar：JFreeChart 的演示实例，使用 java-jar jfreechart-1.0.13-demo.jar 命令可以运行该实例，但 JFreeChart 不提供这些演示实例的源代码。

其他如 licence-LGPL.txt 和 README.txt 等都是说明性文档。



可能读者会感到奇怪，JFreeChart 怎么没有提供入门指南，参考手册等文档呢？因为 JFreeChart 与 JasperReports 项目的策略相似，它们都是使用该项目免费，但文档需要收费的，所以没有提供。

(3) 将 lib 路径下的所有 JAR 文件复制到需要使用 JFreeChart 项目应用的 CLASSPATH 路径下。如果是 Web 应用则需要使用 JFreeChart 图表，即将 lib 下的所有 JAR 文件复制到 Web 应用的 WEB-INF/lib 路径下；如果编译和运行中需要使用 JFreeChart 项目，则还应该将 lib 路径下的 jfreechart-1.0.13.jar 文件添加到系统的 CLASSPATH 环境变量中；如果使用 ANT，或其他 IDE 工具，则无须添加环境变量。

经过上面的三个步骤，则可完成 JFreeChart 的安装。

2. JFreeChart开发

经过上面为使用 JFreeChart 做的准备工作，不难发现通过 JFreeChart 开发统计表其实非常简单。基本上，按如下步骤，即可非常简单地开发出各种统计表。

(1) 提供一个 Dataset 实例，该实例里包含了创建统计图表的数据。



不同的统计图表所使用的 Dataset 实例不一样，例如，饼图使用 PieDataset 实例作为饼图数据；柱状图则采用 CategoryDataset 作为柱状图数据。

(2) 使用 ChartFactory 的多个工厂方法 createXxxChart()来创建统计图表，统计图表就是一个 JFreeChart 对象。

(3) 得到了 JFreeChart 对象后，可以调用 setTitle 来修改统计图表的标题，或者调用 getLegend()方法来获得指定索引的图表图例，取得图例对象后即可修改图表的图例。

(4) 通过 JFreeChart 对象的 getPlot()方法，即可获得图表的 Plot 对象。该对象对应于统计图表的实际图表部分，可以调用 Plot 对象的方法来修改图表中的各种显示内容。

按上面的四个步骤进行，即可快速开发出各种简单的 JFreeChart 图表。

在上面步骤中，共涉及 JFreeChart 的如表 12-1 所示的核心 API。

表 12-1 JFreeChart核心API

类	描述
XxxDataset	这是一个数据集对象，用于提供显示图表所用的数据。对于不同类型的统计图表，也需要使用不同的 Dataset 对象
ChartFactory	该对象是一个图表工厂类，通过调用该工厂类的不同方法，即可生成不同的统计图表
JFreeChart	统计图表对象，JFreeChart 由如下三个部分组成：TextTitle(标题)、LegendTitle(图例标题)和 XxxPlot(实际统计图)
XxxPlot	实际统计图表对象，这个对象决定了实际图表的显示样式，创建该对象时需要 Axis、Renderer 及数据集对象的支持

实际统计图表是根据 XxxPlot 对象生成的，为了生成 XxxPlot 对象，还需要额外的 Axis、Rendererer 的支持。除此之外，如果希望在页面上生成带交互功能的热点图表，还需要用到 XxxURLGenerator 和 XxxToolTipGenerator 等 API。即 JFreeChart 还包含如表 12-2 所示的常用的核心 API。

表 12-2 JFreeChart核心API

类	描述
XxxAxis	用于处理图表的两个轴：纵轴和横轴
XxxRenderer	负责如何显示一个图表对象
XxxURLGenerator	用于生成 Web 图表中每个项目的超级链接
XxxToolTipGenerator	用于生成图表的帮助提示，不同类型图表对应不同类型的工具提示类

12.1.2 实例描述

JFreeChart 为我们提供了生成各类图表的方便，从而能够使我们轻松地将一些复杂的数据利用图表的方式反馈给用户。在上文中已经向大家详细介绍了 JFreeChart 的安装以及开发的方法。在本案例中，将利用一个“机电销售统计图”应用实例向大家讲解 JFreeChart 在实际开发中的使用方法。

12.1.3 实例应用

【例 12-1】 使用 JFreeChart 生成“机电销售统计图”。

(1) 新建 Web Project 项目，解压 jfreechart-1.0.13.zip 压缩包，把它里面 lib 文件夹下的 jfreechart-1.0.13.jar 文件和 jcommon-1.0.16.jar 文件引入到新建项目的 WEB-INF/lib 下。

(2)　在项目中创建 PieChartDemo.java 类，这个类用于生成饼状图。代码如下。

```
package com.struts2.jfreechart;
import java.awt.Font;
import java.io.FileOutputStream;
import java.io.IOException;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PiePlot;
import org.jfree.chart.title.LegendTitle;
import org.jfree.chart.title.TextTitle;
import org.jfree.data.general.DefaultPieDataset;


public class PieChartDemo {
    //创建一个 Dataset 对象，获取数据
    private static DefaultPieDataset getDataSet(){
        DefaultPieDataset dataset=new DefaultPieDataset();
        dataset.setValue("联想笔记本电脑 G450",2000);
        dataset.setValue("索尼手机",1900);
        dataset.setValue("诺基亚手机",1980);
        dataset.setValue("戴尔笔记本",1230);
        return dataset;
    }
    public static void main(String[] args) throws IOException{
        DefaultPieDataset dataset=getDataSet();
        JFreeChart chart=
            ChartFactory.createPieChart(
         "机电销量统计图",//图表标题
         getDataSet(),//数据
         true,//是否显示图例
         false,//是否显示工具提示
         false//是否生成 URL
        );
        //重新设置图表标题，改变字体
        chart.setTitle(new TextTitle("机电销量统计图",new Font("黑体
",Font.ITALIC,22)));
        //取得统计图表的第一个图例
        LegendTitle legendTitle=chart.getLegend(0);
        //修改图例的字体
        legendTitle.setItemFont(new Font("宋体",Font.BOLD,14));
        //获得饼状图的 Plot 对象
        PiePlot plot=(PiePlot)chart.getPlot();
        //设置饼状图各部分的标签字体
        plot.setLabelFont(new Font("隶书",Font.BOLD,18));
        //设置背景透明度(0-1.0 之间)
        plot.setBackgroundAlpha(0.9f);
        //设置前景透明度
        plot.setForegroundAlpha(0.5f);
        //创建一个文件输出流
```

```
        FileOutputStream fos=new FileOutputStream("piechart.jpg");
        //使用 ChartUtilities 将图表输出到文件中
        ChartUtilities.writeChartAsJPEG(
                fos, //输出到哪个输出流
                1, //JPEG 图片的质量，0-1 之间
                chart,//统计图表对象
                800,//宽
                600,//高
                null//ChartRenderingInfo 信息
        );
        fos.close();

    }
}
```

上面代码先提供一个方法，该方法返回 Dataset 对象，这个 Dataset 就是创建统计图表的底层数据，然后调用 ChartFactory 的 createPieChart 方法来生成一个 JFreeChart 对象，这个对象就是统计图表，该图表可以直接输出到图片文件中，也可导出成各种格式的文件。上面代码是导出了一个 JPG 格式的图片文件。

结合上面代码部分可以看出，修改统计图表的标题部分(包括修改图表标题内容、字体大小等)都是通过 JFreeChart 对象的 setTitle()方法实现的，修改统计图表的图例则通过 LegendTitle 对象来完成。一个统计图可以包含多个图例，当调用 JFreeChart 对象的 getLegend(int index)方法时，就可以取得该图表的指定索引的图例对象，一旦取得了指定图例，就可以修改图例的文本内容、字体大小等。

12.1.4 运行结果

运行上面程序，将生成一个 piechart.jpg 文件，使用 Windows 图片和传真查看器浏览该图片，将看到如图 12-2 所示的饼状图。

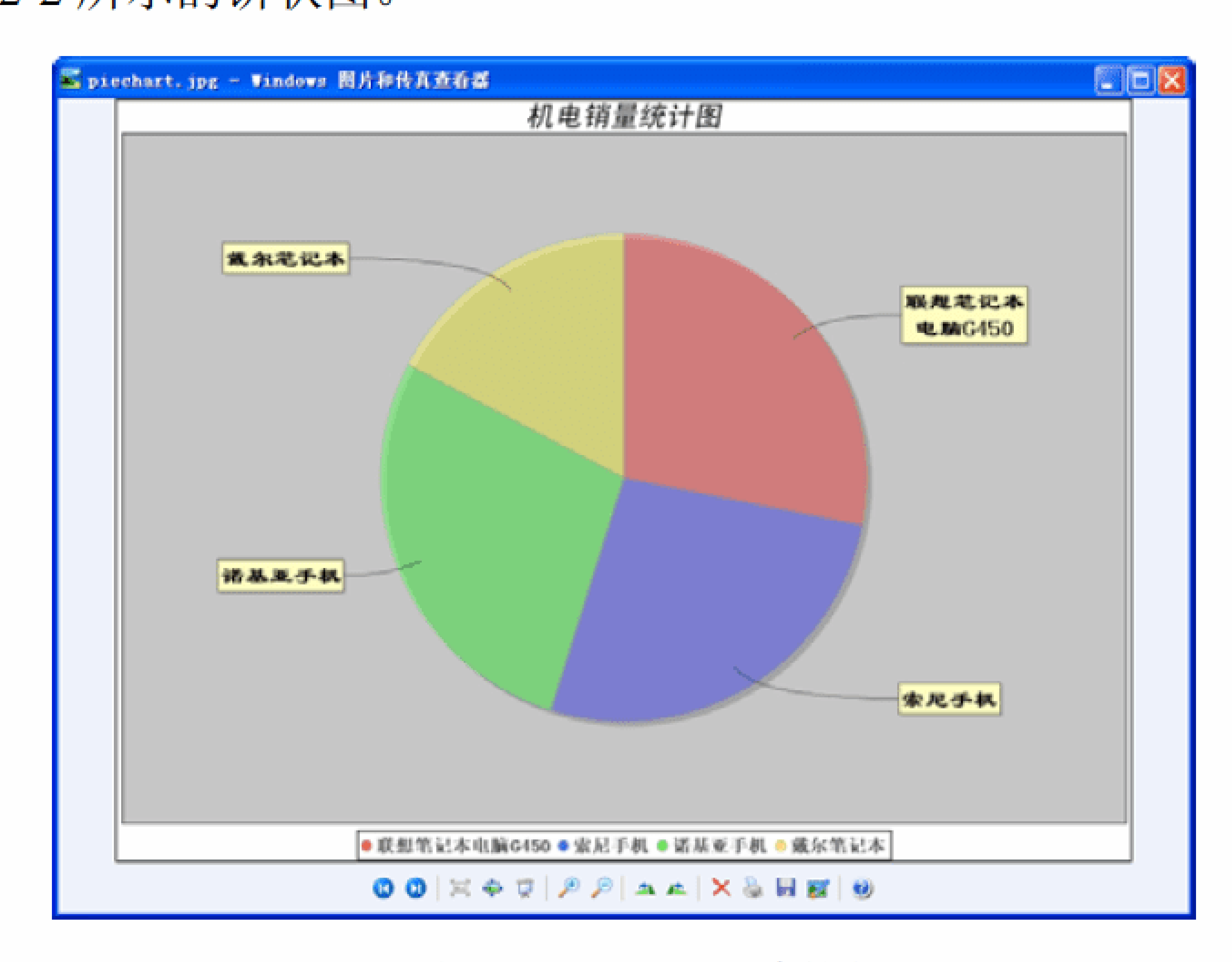


图 12-2 使用JFreeChart生成简单饼状图

12.1.5　实例分析



源码解析：

通过上面的案例，不难发现 JFreeChart 的使用非常简单，只需要提供满足 JFreeChart 要求的数据，即可使用 JFreeChart 创建一个 JFreeChart 图表，就如在上案例中调用 ChartFactory 的 createPieChart 方法，并把数据作为 createPieChart 的一个参数提供给 JFreeChart，从而生成一张漂亮的饼状图。

12.2　JFreeChart统计图表——柱状图

使用 JFreeChart 生成柱状图需要使用 CategoryDataset 作为统计图表的数据载体，生成柱状图后依然可以使用 JFreeChart 来设置统计图表的标题和图例格式。

这一节将为读者介绍使用 JFreeChart 如何生成柱状图。



视频教学： 光盘/videos/12/JFreeChartExample.avi　　　　**长度：** 6 分钟

12.2.1　基础知识——使用JFreeChart生成柱状图

JFreeChart 可以非常方便地生成各种形式的统计图表，包括柱状图。柱状图的 DataSet 一般是用 CatagoryDataset 接口(具体实现类是 DefaultCategoryDataset)，有时也会用 IntervalXYDataset 接口。获取柱状图的 DataSet 对象代码如下所示。

```
    DefaultCategoryDataset defaultcategorydataset = new
DefaultCategoryDataset();
```

其中 DefaultCategoryDataset 类中有一个 addValue()方法，这个方法有如下三个参数。

- double value：数据值。
- java.lang.Comparable rowKey：横轴标签内容。
- java.lang.Comparable columnKey：纵轴标签内容。

这样，就可以为 JFreeChart 所要生成的柱状图创建数据源了，代码如下。

```
private static CategoryDataset getDataset(){
        DefaultCategoryDataset dataset=new DefaultCategoryDataset();
        dataset.setValue(2000,"一月份","联想笔记本电脑 G450");
        dataset.setValue(1900,"一月份","索尼手机");
        dataset.setValue(1980,"一月份","诺基亚手机");
        dataset.setValue(1230,"一月份","戴尔笔记本");
        return dataset;
    }
```

JFreeChart 生成的统计报表图有了数据源，接下来就该对柱状图进行以下设置并加载数据。

首先使用 ChartFactory 的多个工厂方法 createBarChart3D()来创建图表，统计图表是一个 JFreeChart 对象，如下面代码所示。

```
JFreeChart chart=ChartFactory.createBarChart(
                "机电销量统计图", //图表标题
                "各种电器", //目录轴的显示标签
                "销量",//数值轴的显示标签
                getDataset(),//数据集
                //PlotOrientation.HORIZONTAL,//图表方向：水平
                PlotOrientation.VERTICAL,//图表方向：水平 ，垂直
                false,//是否显示图例，对于简单的柱状图必须是 false
                false,//是否生成工具
                false//是否生成 URL 连接
);
```

得到 JFreeChart 对象后，可以调用它的一些方法设置图表的图例。接着调用 JFreeChart 的 getPlot()方法来取得实际图表实例。代码如下。

```
CategoryPlot plot=(CategoryPlot)chart.getPlot();
```

既然是柱状图，就不能像饼状图那样无横、纵轴。获取柱状图的横轴需要调用 CategoryPlot 类中的 getDomainAxis()方法。这个方法得到的是一个 CategoryAxis 类对象，如下所示。

```
    //获取横轴
    CategoryAxis categoryAxis=plot.getDomainAxis();
```

获取柱状图的横轴后，可以调用 CategoryAxis 的 setLableFont(java.awt.Font font)方法来设置横轴显示标签的字体……

对于获取柱状图的纵轴则需要调用 CategoryPlot 类的 getRangeAxis()方法，这个方法得到的是一个 NumberAxis 对象，如下所示。

```
    //获取纵轴
    NumberAxis numberAxis=(NumberAxis)plot.getRangeAxis();
```

在 NumberAxis 类中也存在 setLabelFont(java.awt.Font font)方法，用来设置纵轴显示标签的字体。

到此为止，对柱状图的图例格式就设置完毕了，最后将统计图表输出成 JPG 文件！代码如下。

```
FileOutputStream fos=new FileOutputStream("categoryChart.jpg");
    //将柱状图表输出成 JPG 文件
    ChartUtilities.writeChartAsJPEG(
            fos,//输出到哪个输出流
            1, //jpeg 图片的质量
            chart,//统计图表对象
            800,//宽
            600,//高
            null//ChartRenderingInfo 信息
            );
fos.close();
```

12.2.2　实例描述

我那个卖机电的朋友事真多，今天又上门拜访我，让我把他们公司各个产品的销售金额做成一张柱状图的图片。真是烦啊，谁让我答应他第一次来着，让他抓住了我的“小把柄”了，让他知道我会用代码生成一张他想要的图片了，这不他三番五次的来找我，我算是服了他了，不过没办法，谁让他是我朋友呢。

这次又是三下五除二，不费吹灰之力帮他解决了这个难题。

12.2.3　实例应用

【例 12-2】 使用 JFreeChart 生成柱状图。

在项目中创建生成柱状图的类 BarChartDemo.java，代码如下。

```
package com.struts2.jfreechart;

import java.awt.Font;
import java.io.FileOutputStream;
import java.io.IOException;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.CategoryAxis;
import org.jfree.chart.axis.CategoryLabelPositions;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.title.LegendTitle;
import org.jfree.chart.title.TextTitle;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;
public class BarChartDemo {
    private static CategoryDataset createDataset()
    {
    String series1 = "January";
    String series2 = "February";
    String series3 = "March";
    String category1 = "联想笔记本电脑 G450";
    String category2 = "索尼手机";
    String category3 = "诺基亚手机";
    String category4 = "戴尔笔记本";
    String category5 = "松下血压仪";
    DefaultCategoryDataset defaultcategorydataset = new
DefaultCategoryDataset();
    defaultcategorydataset.addValue(2000D, series1, category1);
    defaultcategorydataset.addValue(4000D, series1, category2);
```

```
    defaultcategorydataset.addValue(3000D, series1, category3);
    defaultcategorydataset.addValue(5000D, series1, category4);
    defaultcategorydataset.addValue(5120D, series1, category5);
    defaultcategorydataset.addValue(5441D, series2, category1);
    defaultcategorydataset.addValue(7111D, series2, category2);
    defaultcategorydataset.addValue(6000D, series2, category3);
    defaultcategorydataset.addValue(8200D, series2, category4);
    defaultcategorydataset.addValue(4000D, series2, category5);
    defaultcategorydataset.addValue(4125D, series3, category1);
    defaultcategorydataset.addValue(3411D, series3, category2);
    defaultcategorydataset.addValue(2141D, series3, category3);
    defaultcategorydataset.addValue(3541D, series3, category4);
    defaultcategorydataset.addValue(6000D, series3, category5);
    return defaultcategorydataset;

    }
    public static void main(String[] args) throws IOException{
        JFreeChart jfreechart = ChartFactory.createBarChart(
                "机电销量统计图", //图形标题名称
                "销量",//目录横轴的显示标签
                "总额", //数值轴的显示标签
                createDataset(), // dataset
                PlotOrientation.VERTICAL, //垂直显示
                true, // 显示图例
                true, // 生成工具
                false); //不要生成 URL 连接
        //重新设置图表标题，改变字体
        jfreechart.setTitle(new TextTitle("机电销售统计图",new Font("黑体
",Font.ITALIC,22)));
        //取得统计图表的第一个图例
        LegendTitle legendTitle=jfreechart.getLegend(0);
        //修改图例的字体
        legendTitle.setItemFont(new Font("宋体",Font.BOLD,14));
        CategoryPlot plot=(CategoryPlot)jfreechart.getPlot();
        //取得横轴
        CategoryAxis categoryAxis=plot.getDomainAxis();
        //设置横轴显示标签的字体
        categoryAxis.setLabelFont(new Font("宋体",Font.BOLD,22));
        //分类标签以 45 度角倾斜

    categoryAxis.setCategoryLabelPositions(CategoryLabelPositions.UP_45);
        categoryAxis.setTickLabelFont(new Font("宋体",Font.BOLD,18));
        //取得纵轴
        NumberAxis numberAxis=(NumberAxis)plot.getRangeAxis();
        //设置纵轴显示标签的字体
        numberAxis.setLabelFont(new Font("宋体",Font.BOLD,22));
        //创建一个文件输出流
        FileOutputStream fos=new FileOutputStream("barchart.jpg");
        //使用 ChartUtilities 将图表输出到文件中
```

```
        ChartUtilities.writeChartAsJPEG(
                        fos, //输出到哪个输出流
                        1, //JPEG 图片的质量，0-1 之间
                        jfreechart,//统计图表对象
                        800,//宽
                        600,//高
                        null//ChartRenderingInfo 信息
        );
        fos.close();
    }
}
```

在上面的代码中，为了修改柱状图坐标轴的显示格式，我们使用了两个 XxxAxis 实例，其中 CategoryAxis 代表柱状图的横轴， NumberAxis 代表柱状图的纵轴。分别调用 CategoryPlot 的如下两个方法，即可获得柱状图的横轴和纵轴。

- getDomainAxis：返回柱状图的横轴。
- getRangeAxis：返回柱状图的纵轴。

12.2.4　运行结果

运行 BarChartDemo 类，在本项目的根路径下生成一张名为 barchart.jpg 的统计图，如图 12-3 所示。

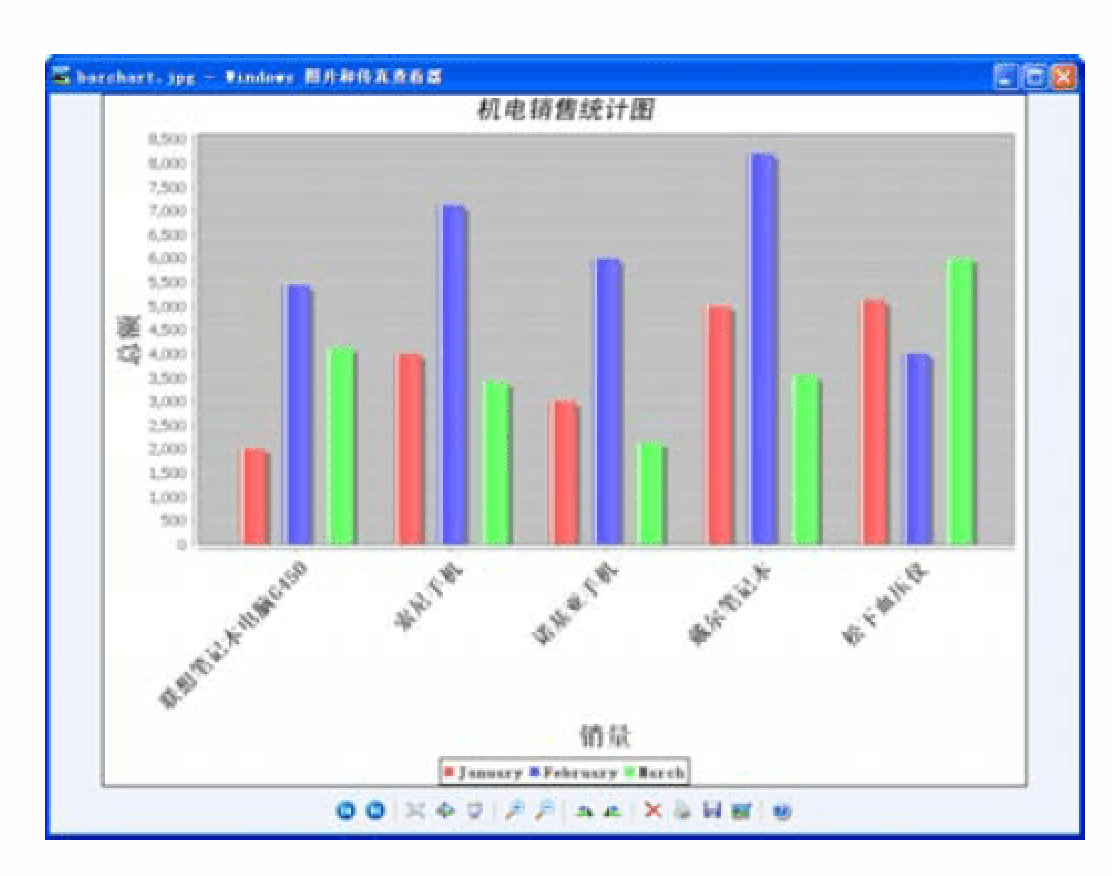
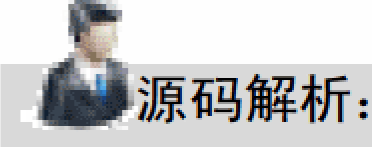


图 12-3　使用JFreeChart生成柱状图

12.2.5　实例分析

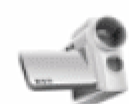


源码解析：

在上案例中，我们创建了 CategoryDataset 实例，该实例调用 addValue()方法时，传入了三个参数，其中第二个参数是一组数据的 key。因为该程序使用了不同的 key，所以程序在统计图表中使用了图例，并且修改了图例的格式。

12.3 JFreeChart统计图表——折线图

对于数据的变化、发展情况或发展趋势可以使用折线图来描述，其显示方式非常直观。本节将为读者讲解一下使用 JFreeChart 组件如何生成折线图。

视频教学：光盘/videos/12/JFreeChartExample.avi　　长度：6 分钟

12.3.1 基础知识——使用JFreeChart生成折线图

因为折线图和柱状图的图形结构大致相似，区别在于柱状图使用立方体(或者矩形)表示数值，而折线图则采用固定值表示数值。因此，生成折线图与生成柱状图的过程大致相似，同样可以采用 CategoryDataset 实例作为统计图表的数据。

按照 JFreeChart 的开发步骤第二步，有了数据源，就可以使用 ChartFactory 的多个工程方法 createXxxChart 来创建统计图表，折线图调用了 ChartFactory 的 createLineChart3D 这个方法来创建 JFreeChart 对象。如下面代码所示。

```
JFreeChart chart = ChartFactory.createLineChart3D(
        "2004-2010 年优秀杀毒软件杀毒数量统计",//图表标题
        "杀毒软件",//X 轴标题
        "查杀病毒数量",//Y 轴标题
        createDataSet(),//绘图数据集，调用了获取数据源的方法
        PlotOrientation.VERTICAL,//绘制方向
        true, //显示图例
        true, //采用标准生成器
        false//是否生成超链接
);
```

前面一节讲到的使用 JFreeChart 生成柱状图代码中，获取绘图区对象，即 CategorypLot 对象采用的方法是调用 JFreeChart 的 getPlot()方法。从 JFreeChart 的 API 文档中可以看出，JFreeChart 类中存在两个方法：getPlot()和 getCategoryPlot()。其中，前者返回的是一个 Plot 对象，它是 CategoryPlot()类的父类，后者返回的直接就是一个 CategoryPlot 对象。因此可以直接调用 JFreeChart 类中的 getCategoryPlot()方法来获取绘图区对象，即

```
CategoryPlot plot = chart.getCategoryPlot();
```

获取绘图区对象后，可以调用它的一些方法来对绘图区进行美化，比如设置背景色、水平方向背景线颜色、垂直方向背景线颜色、横轴字体颜色和内容等。把绘图区域绘制好之后，来对生成的折线图进行一下设置。首先需要获取折线图对象，即

```
    LineAndShapeRenderer renderer = (LineAndShapeRenderer)
plot.getRenderer();
```

折线图有虚线也有实线，设置虚线的代码如下所示。

```
    float dashes[] = { 8.0f };//定义虚线数组
    BasicStroke brokenLine = new BasicStroke(1.6f, //线条粗细
        BasicStroke.CAP_SQUARE, //端点风格
        BasicStroke.JOIN_MITER, //折点风格
        8.f,//折点处理办法
        dashes, //虚线数组
        0.0f//虚线偏移量
    );
```

设置实线代码为。

```
BasicStroke realLine = new BasicStroke(1.6f); //设置实线
```

设置折线图使用实线绘制还是虚线绘制，需要调用 LineAndShapeRenderer 类的 setSeries Stroke(int series,java.awt.Stroke stroke)方法。

这个方法有两个参数，第一个参数表示要绘制的第几条线，是一个以 1 开头的 int 类型的数字；第二个参数是一个 java.awt.Stroke 对象，表示该条线是要用实现绘制还是要用虚线绘制。如下面代码，表示第一条线用虚线绘制。

```
renderer.setSeriesStroke(1, brokenLine);
```

最后使用 ChartUtilities 将图表输出到文件中，生成一张图片。到此，使用 JFreeChart 生成折线图就完成了。

12.3.2　实例描述

我们经理真是天天没事干瞎折腾，这几天他又想知道他电脑上的杀毒软件在最近几年的杀毒情况。我需要声明一下啊，他的电脑已经有 10 年的寿命了，并且电脑上的杀毒软件更是多得数不胜数，也许这就是他的电脑能“活”到现在的最贴切的原因吧！

我们经理比较“器重”我，就让我接了这活。

还好，我的技术是经得起考验的。下面案例就是使用折线图对 2004—2010 年杀毒软件杀毒数量进行统计，其显示效果分为普通样式和 3D 样式。本实例数据集合的数据随机产生，故每次显示的结果并非一致。

12.3.3　实例应用

【例 12-3】 使用折线图统计 2004—2010 年优秀杀毒软件的杀毒情况。

在本章的项目中继续创建 XYLineChart 类，用于生成折线图。内容如下。

```
package com.struts2.jfreechart;
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Font;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Random;
```

```
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.category.LineAndShapeRenderer;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;

public class XYLineChart {
    // 字体
    private static final Font PLOT_FONT = new Font("宋体", Font.BOLD, 15);
    /**
     * 创建数据集合
     *
     * @return CategoryDataset 对象
     */
    public static CategoryDataset createDataSet() {
        // 图例名称
        String[] line = { "杀毒软件一", "杀毒软件二", "杀毒软件三" };
        // 类别
        String[] category = { "2004 年", "2005 年", "2006 年", "2007 年", "2008
年","2009 年","2010 年" };
        Random random = new Random();// 实例化 Random 对象
        // 实例化 DefaultCategoryDataset 对象
        DefaultCategoryDataset dataSet = new DefaultCategoryDataset();
        // 使用循环向数据集合中添加数据
        for (int i = 0; i < line.length; i++) {
            for (int j = 0; j < category.length; j++) {
                dataSet.addValue(100000 + random.nextInt(100000), line[i],
                        category[j]);
            }
        }
        return dataSet;
    }
    /**
     * 生成制图对象
     *
     * @param is3D 是否为 3D 效果
     * @return JFreeChart 对象
     */
    public static void main(String[] args) throws IOException{

        java.util.Scanner input = new java.util.Scanner(System.in);
        System.out.print("是要 3D 效果吗? (true/false): ");
        String is3D = input.next();

        JFreeChart chart = null;
        if(is3D.equals("true")){
```

```
        chart = ChartFactory.createLineChart3D(
                    "2004-2010 年优秀杀毒软件杀毒数量统计",//图表标题
                    "杀毒软件",//X 轴标题
                    "查杀病毒数量",//Y 轴标题
                    createDataSet(),//绘图数据集
                    PlotOrientation.VERTICAL,//绘制方向
                    true,//显示图例
                    true,//采用标准生成器
                    false//是否生成超链接
                    );
    }else{
        chart = ChartFactory.createLineChart(
                    "2004-2010 年优秀杀毒软件杀毒数量统计", //图表标题
                    "杀毒软件",//X 轴标题
                    "查杀病毒数量",//Y 轴标题
                    createDataSet(),
                    //绘图数据集
                    PlotOrientation.VERTICAL,//绘制方向
                    true,//是否显示图例
                    true,//是否采用标准生成器
                    false//是否生成超链接
                    );
        }
        //设置标题字体
        chart.getTitle().setFont(new Font("隶书", Font.BOLD, 23));
        //设置图例类别字体
        chart.getLegend().setItemFont(new Font("宋体", Font.BOLD, 15));
        chart.setBackgroundPaint(new Color(192,228,106));   //设置背景色
        //获取绘图区对象
        CategoryPlot plot = chart.getCategoryPlot();
        plot.getDomainAxis().setLabelFont(PLOT_FONT);//设置横轴字体
        plot.getDomainAxis().setTickLabelFont(PLOT_FONT);//设置坐标轴标尺值字体
        plot.getRangeAxis().setLabelFont(PLOT_FONT);//设置纵轴字体
        plot.setBackgroundPaint(Color.WHITE);//设置绘图区背景色
        plot.setRangeGridlinePaint(Color.RED);//设置水平方向背景线颜色
        plot.setRangeGridlinesVisible(true);//设置是否显示水平方向背景线,默认值为
                                           true
        plot.setDomainGridlinePaint(Color.RED);//设置垂直方向背景线颜色
        plot.setDomainGridlinesVisible(true);//设置是否显示垂直方向背景线,默认值
                                            为 false
        //获取折线对象
        LineAndShapeRenderer renderer = (LineAndShapeRenderer)
plot.getRenderer();
        BasicStroke realLine = new BasicStroke(1.6f);//设置实线
        float dashes[] = { 8.0f };//定义虚线数组
        BasicStroke brokenLine = new BasicStroke(1.6f,//线条粗细
                BasicStroke.CAP_SQUARE,//端点风格
                BasicStroke.JOIN_MITER,//折点风格
                8.f, //折点处理办法
```

```
                dashes, //虚线数组
                0.0f//虚线偏移量
                );
        renderer.setSeriesStroke(1, brokenLine);//利用虚线绘制
        renderer.setSeriesStroke(2, brokenLine);//利用虚线绘制
        renderer.setSeriesStroke(3, realLine);//利用实线绘制
        //创建一个文件输出流
        FileOutputStream fos=new FileOutputStream("linechart.jpg");
        //使用 ChartUtilities 将图表输出到文件中
        ChartUtilities.writeChartAsJPEG(
                            fos, //输出到哪个输出流
                            1, //JPEG 图片的质量，0-1 之间
                            chart,//统计图表对象
                            800,//宽
                            600,//高
                            null//ChartRenderingInfo 信息
        );
        fos.close();
    }
}
```

其中，createDataSet()方法用于创建数据集合对象。实例中使用 JDK 中的 Random 类所产生的随机数据进行填充。它返回 CategoryDataSet 对象。程序的入口函数中要求从控制台输入一个 Boolean 值，根据输入的值来判断生成普通样式折线图还是 3D 样式折线图效果。

12.3.4 运行结果

运行程序，控制台输出如下。

```
是要 3D 效果吗？(true/false):
```

在控制台中输入 true，打开本项目的根目录，生成一张名为 linechart.jpg 的图片，如图 12-4 所示。

再次运行程序，当控制台出现“是要 3D 效果吗？(true/false)：”时，在其后面输入 false，生成普通样式折线图，如图 12-5 所示。



图 12-4　3D样式折线图

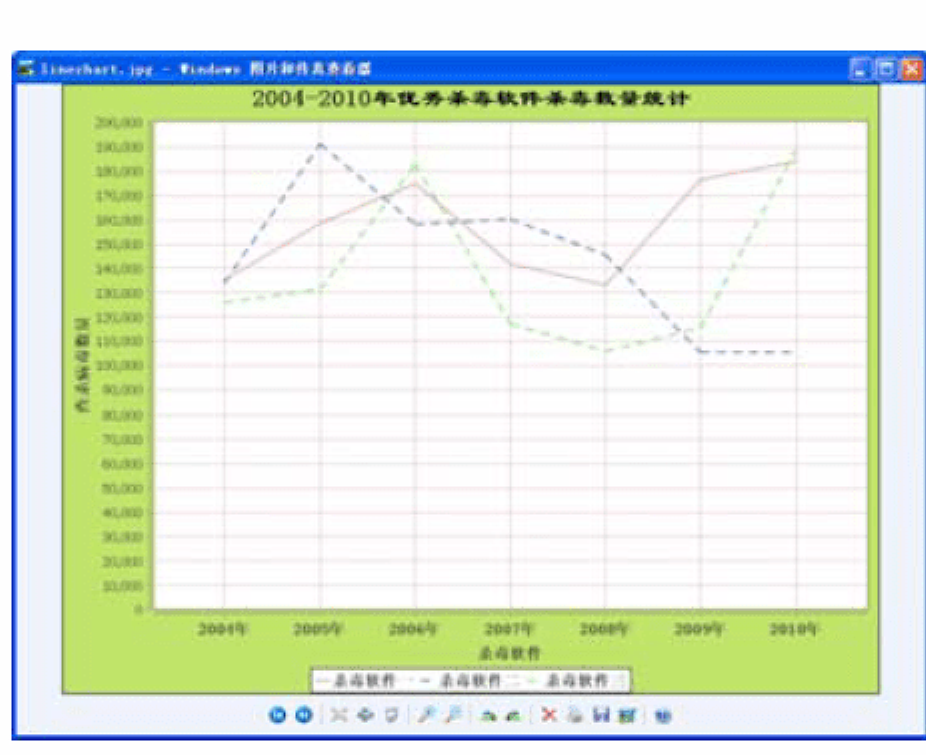


图 12-5　普通样式折线图

12.3.5　实例分析



源码解析：

上案例中，在程序的主函数中要求输入一个 Boolean 值，来判断生成的折线图效果是 3D 样式还是普通样式。从代码的角度来看，不管是 3D 样式还是普通样式并没有太大的区别，只有一点区别，就是调用 ChartFactory 类的 createLineChart3D()方法生成的是 3D 样式，而调用它的 createLineChart()方法生成的就是普通样式，其他在设置折线图的其他特性上都是相同的。

12.4　JFreeChart统计图表——时间顺序图

时间顺序图可用于统计、分析数据随时间变化的发展情况。它的创建需要用到 XYDataset 数据集合。本节介绍使用 JFreeChart 如何生成时间顺序图。



视频教学： 光盘/videos/12/JFreeChartExample.avi　　　　长度：6 分钟

12.4.1　基础知识——使用JFreeChart生成时间顺序图

时间顺序图与上面的折线图非常相似，但区别在于时间顺序图不可使用 3D 图形，且时间顺序图的水平坐标轴是时间。

生成时间顺序图需要使用 XYDataset 实例作为统计图的底层数据。XYDataset 有一个实现类：TimeSeriesCollection。

这个时间序列实例是由多个 TimeSeries 对象组成的，每个 TimeSeries 包含多个时间点的统计值。通过 TimeSeriesCollection 将多个 TimeSeries 对象组合起来，就可以生成所需要的时间顺序图。

使用 JFreeChart 来生成时间顺序图，与生成其他统计图并没有太大的差别，只是生成时间顺序图要求使用 XYDataset 作为底层数据，故使用 TimeSeriesCollection 来组合多个 TimeSeries 实例，如下面代码所示。

```
private static XYDataset getDataset(){
        //创建第一个 TimeSeries 实例
        TimeSeries ts1=new TimeSeries("杀毒软件一");
        //为第一个 TimeSeries 添加不同时间点的统计值
        ts1.add(new Month(10,2009),3400);
        /*可以填充更多时间点的统计值*/
        //创建第二个 TimeSeries 实例
        TimeSeries ts2=new TimeSeries("杀毒软件二");
        //为第二个 TimeSeries 添加不同时间点的统计值
        ts2.add(new Month(10,2009),2800);
```

```
        /*可以填充更多时间点的统计值*/
        //创建 TimeSeriesCollection 实例
        TimeSeriesCollection datasetCollection=new TimeSeriesCollection();
        //使用 TimeSeriesCollection 来组合多个 TimeSeries 实例
        datasetCollection.addSeries(ts1);
        datasetCollection.addSeries(ts2);
        return datasetCollection;
    }
```

该 TimeSeriesCollection 就是一个 XYDataset 实例。接下来使用 ChartFactory 类中的 create TimeSeriesChart()方法创建统计图表，即如下代码。

```
JFreeChart chart=ChartFactory.createTimeSeriesChart(
        "2004-2010 年优秀杀毒软件杀毒数量统计",//图表标题
        "杀毒软件",//X 轴标题
        "查杀病毒数量",//Y 轴标题
        getDataset(),//绘图数据集
        true,//显示图例
        true,//采用标准生成器
        false//是否生成超链接
);
```

获取了统计图表实例，可以调用它的一些方法对生成的时间顺序图进行一些勾勒，和前面所讲到的使用 JFreeChart 生成的统计图表代码大同小异，这里不再累赘。值得注意的是，取得时间顺序图的 Plot 对象时需要调用 JFreeChart 类中的 getXYPlot()方法，即如下代码。

```
XYPlot plot=chart.getXYPlot();
```

12.4.2 实例描述

今天闲来无事，就看一本《Java 基础教程》的书，这本书的销售额之高令人惊呼。好奇心的我很想知道这本书今年的销售量是什么样的情况？当然，我也不知道这本书在每个月的销售数量达到多少，数据只能是随机生成。

本实例中，使用 TimeSeries 对象填充 10 个月的数据，然后通过时序图统计书的销售量。本实例数据并非是真实数据，是随机而生，因此每次显示结果并非一致。

12.4.3 实例应用

【例 12-4】 通过时序图统计书的销售量。

在本章项目中创建自定义制图工具类 ChartUtil，在此类中编写两个方法，分别用于创建数据集合及生成时间顺序图。其关键代码如下。

```
public class ChartUtil {
    //字体
    private static final Font PLOT_FONT = new Font("黑体", Font.ITALIC , 18);
    //返回一个 XYDataset 实例
```

```
    private static XYDataset getDataset(){
        //实例化 TimeSeries 对象
        TimeSeries timeseries = new TimeSeries("Data");
        Day day = new Day(1, 1, 2010);            //实例化 Day
        double d = 3000D;                      //添加一年 365 天的数据
        for (int i = 0; i < 305; i++) {
            d = d + (Math.random() - 0.5) * 10;         //创建随机数据
            timeseries.add(day, d);                    //向数据集合中添加数据
            day = (Day) day.next();
            }
        //创建 TimeSeriesCollection 集合对象
        TimeSeriesCollection timeSeriesCollection =new
TimeSeriesCollection(timeseries);
        //返回数据集合对象
        return timeSeriesCollection;
        }
    public static void main(String[] args) throws IOException{
        //创建时序图对象
        JFreeChart chart = ChartFactory.createTimeSeriesChart(
                "Java 基础全国销量统计",        //标题
                "销售月份",               //时间轴标签
                "销量(本)",               //数据轴标签
                getDataset(),              //数据集合
                false,                     //是否显示图例标识
                false,                     //是否显示 tooltips
                false);                //是否支持超链接
        //设置标题字体
        chart.getTitle().setFont(new Font("隶书", Font.BOLD, 26));
        //设置背景色
        chart.setBackgroundPaint(new Color(252,175,134));
        XYPlot plot = chart.getXYPlot();               //获取图表的绘制属性
        plot.setDomainGridlinesVisible(false);           //设置网格不显示
        //获取时间轴对象
        DateAxis dateAxis = (DateAxis) plot.getDomainAxis();
        dateAxis.setLabelFont(PLOT_FONT);              //设置时间轴字体
        //设置时间轴标尺值字体
        dateAxis.setTickLabelFont(new Font("宋体",Font.PLAIN,12));
        dateAxis.setLowerMargin(0.0);              //设置时间轴上显示的最小值
        //获取数据轴对象
        ValueAxis valueAxis = plot.getRangeAxis();
        valueAxis.setLabelFont(PLOT_FONT);                //设置数据字体
        DateFormat format = new SimpleDateFormat("MM 月份"); //创建日期格式对象
        //创建 DateTickUnit 对象
        DateTickUnit dtu = new DateTickUnit(DateTickUnitType.DAY,29,format);
        dateAxis.setTickUnit(dtu);                     //设置日期轴的日期标签
        //创建一个文件输出流
        FileOutputStream fos=new FileOutputStream("timeSeries.jpg");
        //使用 ChartUtilities 将图表输出到文件中
        ChartUtilities.writeChartAsJPEG(
                                fos,                   //输出到哪个输出流
                                1,                     //JPEG 图片的质量，0-1 之间
```

```
                    chart,                //统计图表对象
                    800,                  //宽
                    600,                  //高
                    null                  //ChartRenderingInfo 信息
            );
            fos.close();
        }
}
```

此类中属性 PLOT_FONT 是一个静态的字体常量对象，使用此对象可以避免反复用到的字体对象被多次创建。

getDataset()方法用于创建数据集合对象。时间顺序图的数据集合与其他数据集合不同，它需要添加一个时间段内的所有数据，通常采用 TimeSeries 类进行添加。由于数据量较大，实例中通过 Math 类的 random()方法进行随机生成。

程序的主函数用于创建制图对象，它返回 JFreeChart 对象。由于时间顺序图属于坐标轴类型的图表，实例中通过日期轴对象 DateAxis 与时间轴对象 ValueAxis 对相关属性进行设置。

12.4.4 运行结果

运行程序，生成一张名为 timeSeries.jpg 的图片，如图 12-6 所示。

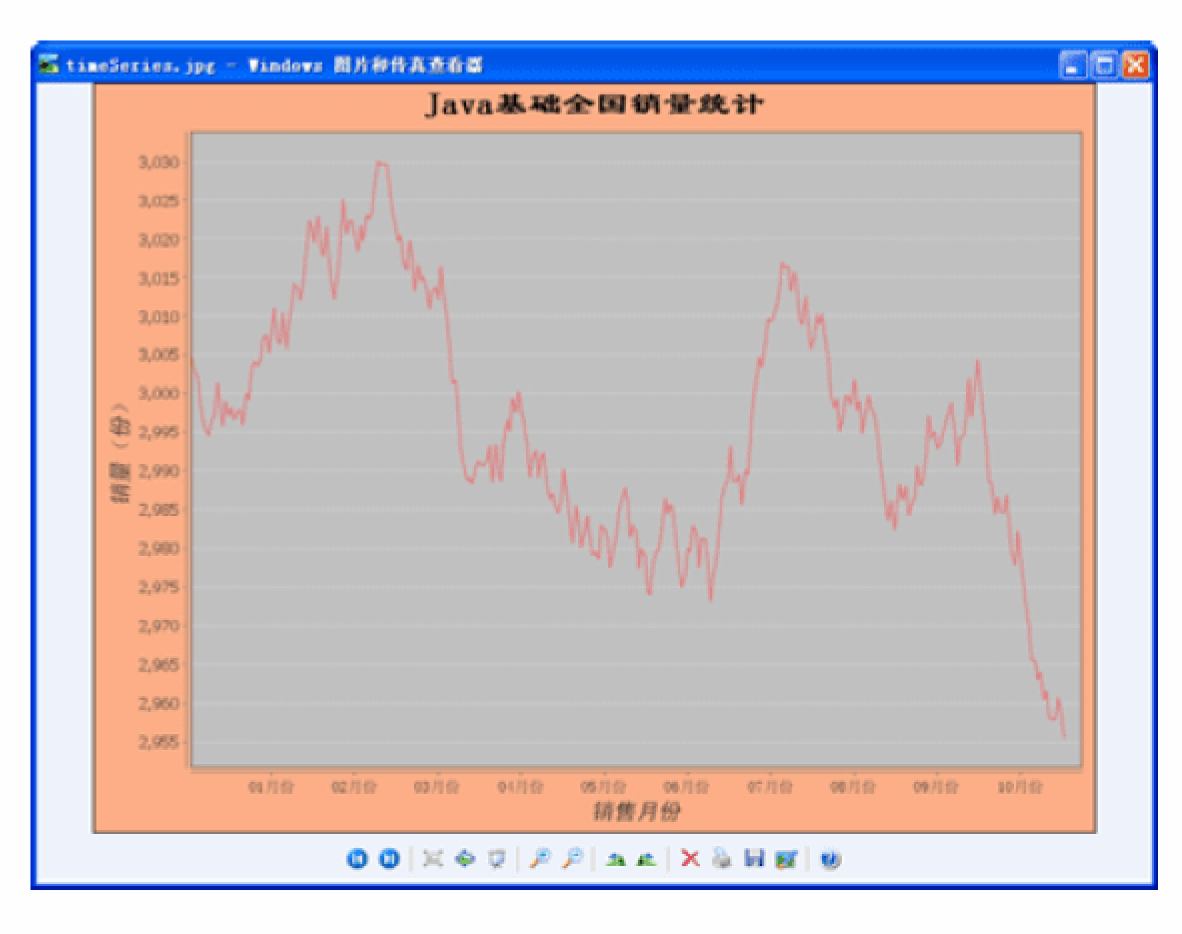
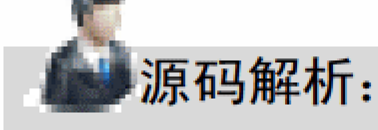


图 12-6 使用JFreeChart生成时间顺序图

12.4.5 实例分析

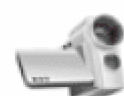

源码解析：

通过上面的案例，不难发现：使用 JFreeChart 来生成各种统计图表是相当简单的事情，只要按前面所指示的步骤，即可生成各种统计图表。在生成不同的统计图表的过程中，关键差别在于需要使用不同的 Dataset 作为底层数据源。

12.5　在网页中生成带交互功能的统计图

很多时候，需要直接在 JSP 页面中生成统计图表。当需要在 JSP 页面中生成统计图表时，常用的做法就是提供系列的自定义标签。自定义标签封装了 JFreeChart 的统计图功能，从而简化 JFreeChart 的开发。

除此之外，还可以利用网页图片的热点功能，生成带交互功能的统计图。本节将为读者介绍如何使用 JFreeChart 在网页中生成带交互功能的统计图。

 **视频教学：**光盘/videos/12/JFreeChartExample.avi　　　　长度：6 分钟

12.5.1　基础知识——在网页中生成带交互功能的统计图

生成带交互功能的统计图需要使用 JFreeChart 项目的三个核心 API，分别是 XxxToolTipGenerator、XxxURLGenerator 和 ChartRendererInfo，关于它们的功能简介如下。

- XxxToolTipGenerator：用于生成图表的帮助提示，当鼠标在统计图的各部分悬停时，系统显示提示信息。
- XxxURLGenerator：用户生成 Web 图表中每个项目的超级连接。
- ChartRendererInfo：该对象封装了统计图表显示的相关信息。

繁多的理论知识不如来了实例“吸收”的快。下面就以一段代码来演示一下如何运用上面三个核心 API 在网页中生成带交互功能的统计图。

```
        //创建 3D 饼图的 Plot 对象
        PiePlot3D plot3D=new PiePlot3D(dataset);
        //生成 3D 饼图的图表
        JFreeChart chart=new
JFreeChart("",JFreeChart.DEFAULT_TITLE_FONT,plot3D,true);
        //生成饼图各部分的提示，当鼠标悬停时显示实际统计值
        plot3D.setToolTipGenerator(new StandardPieToolTipGenerator());
        //设定热点链接
        plot3D.setURLGenerator(new StandardPieURLGenerator("piechart.jsp"));
        StandardEntityCollection entityCollection=new
StandardEntityCollection();
        //生成 RenderingInfo 实例
        ChartRenderingInfo info=new ChartRenderingInfo(entityCollection);
        //在 Web 服务器的临时目录生成一张图片，800 是图片长度，600 是图片高度
        //将图表的热点信息在 HTML 页面中输出，pw 代表页面的输出流，map 是定义热点的 Map 标签 ID
        //info 参数就是图片的热点信息
        ChartUtilities.writeImageMap(pwPrintWriter, "map", info, false);
```

在上面代码中使用了一个 ServletUtilities 类，这个类与前面介绍的 ChartUtilities 功能大致相似，都用于输出各种格式的图片文件。它们区别在于 ChartUtilities 通常用于直接将报表图片

输出到指定图片文件中，但 ServletUtilities 则将报表图片输出到 Web 服务器的临时目录下，并且无需指定图片文件的文件名，而是由系统自动生成该图片文件的文件名。

上面的代码可以生成一张统计图表，这张统计图表图片被放在了 Web 服务器的临时目录下。为了提供更好的解耦，让 Web 应用与不同的 Web 服务器解耦，JFreeChart 提供了一个名为“DisplayChart”的 Servlet，这个 Servlet 专门用于显示不同 Web 服务器的临时目录下的统计图片。



在 Web 应用中使用 JFreeChart 项目时，应该使用 ServletUtilities 来生成统计图片，ServletUtilities 工具类会自动将统计图片放入 Web 服务器的临时目录下，然后通过 DisplayChart 工具 Servlet 来显示 Web 服务器临时目录下的指定图片。

为了使用 DisplayChart Servlet 来显示 Web 服务器临时目录下的图片，应该在 Web 应用的 web.xml 文件中配置 DisplayChart Servlet，因此应该在 web.xml 文件中增加如下片段。

```
<servlet>
    <servlet-name>DisplayChart</servlet-name>
    <servlet-class>org.jfree.chart.servlet.DisplayChart</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>DisplayChart</servlet-name>
    <url-pattern>/DisplayChart</url-pattern>
</servlet-mapping>
```

在 JSP 页面中使用 DisplayChart 来显示 Web 服务器端临时目录下的统计图片，即带热点交互功能的HTML 标签代码如下所示。

```
<img src="DisplayChart" width="720" height="450" usemap="#map">
```



usemap 完全是一个 HTML 标签属性，它用于将 HTML 页面上的图片划分出几个区域，每个区域对应一个热点，每个热点可单独指定一个超级链接。

在浏览器中浏览该 JSP 页面后，表面上看起来与前面介绍的 3D 饼图并没有太大的差别。但当把鼠标移动到饼图的任何一个部分时，可看到光标变成手的形状。这表明该图片的这个部分包含了热点链接。在左下角的地址栏中，也可以看到该链接的 URL 信息。实际上，该链接的 URL 中包含了光标所在饼图部分的信息。

上面代码使用 JFreeChart 核心 API 中的 XxxURLGenerator()方法，指定了所要对应的 URL 为 piechart.jsp 页面，这就是该统计图上热点链接的地址。

如此编写代码，当点击饼图中的任何部分时都会链接到 piechart.jsp 页面，从而实现在页面中生成带交互功能的统计图。

12.5.2 实例描述

从一张统计图上只能看出该产品的销售量在总销售量中所占的比例，但无法比较该产品的销售量在本类产品总销售量中所占的比例。上次为我朋友做的销量统计图只能实现宏观的观看某产品在总销售量中所占的比例，朋友公司老总很是不满，这次又让我给他改动一下，改成在

网页上生成饼状统计图，并点击某一部分时要跳转至该产品的销售量在该类产品的总销售量中所占比例的统计图页面。哈哈……一名优秀的开发者是经得起“风吹雨打”的。

12.5.3　实例应用

【例 12-5】 使用 JFreeChart 在网页中生成带交互功能的“机电销量统计图”。

(1) 在 WebRoot 下创建生成带热点交互功能的统计图的 JSP 页面，内容片段如下。

```
<%@ page contentType="text/html;charset=GBK"%>
<%@ page import="org.jfree.data.general.DefaultPieDataset"%>
<%@ page import="org.jfree.chart.*"%>
<%@ page import="org.jfree.chart.plot.*"%>
<%@ page import="org.jfree.chart.servlet.ServletUtilities"%>
<%@ page import="org.jfree.chart.urls.StandardPieURLGenerator"%>
<%@ page import="org.jfree.chart.entity.StandardEntityCollection"%>
<%@ page import="java.io.*,java.awt.*"%>
<%@ page import="org.jfree.chart.labels.*"%>
<%@ page import="org.jfree.chart.title.*"%>
<%
        //创建饼图所需的 DefaultPieDateset 数据
        DefaultPieDataset dataset=new DefaultPieDataset();
        dataset.setValue("联想笔记本电脑 G450",2000);
        dataset.setValue("索尼手机",1900);
        dataset.setValue("诺基亚手机",1980);
        dataset.setValue("戴尔笔记本",1230);
        //创建 3D 饼图的 Plot 对象
        PiePlot3D plot3D=new PiePlot3D(dataset);
        plot3D.setLabelFont(new Font("隶书",Font.BOLD,16));
        //生成 3D 饼图的图表
        JFreeChart chart=new
JFreeChart("",JFreeChart.DEFAULT_TITLE_FONT,plot3D,true);
        //重新设置图表标题，改变字体
        chart.setTitle(new TextTitle("机电销量统计图",new Font("黑体
",Font.ITALIC,22)));
        //获取统计图表的图例对象
        LegendTitle legendTitle=chart.getLegend(0);
        //修改图例的字体
        legendTitle.setItemFont(new Font("宋体",Font.BOLD,13));
        //生成饼图各部分的提示，当鼠标悬停时显示实际统计值
        plot3D.setToolTipGenerator(new StandardPieToolTipGenerator());
        //设定热点链接
        plot3D.setURLGenerator(new StandardPieURLGenerator("piechart.jsp"));
        StandardEntityCollection entityCollection=new
StandardEntityCollection();
        //生成 RenderingInfo 实例
        ChartRenderingInfo info=new ChartRenderingInfo(entityCollection);
        //创建一个文件输出流
```

```
        //将页面输出流 out 包装成一个 PrintWriter 实例
        PrintWriter pwPrintWriter=new PrintWriter(out);
        //在 Web 服务器的临时目录生成一张图片，800 是图片长度，600 是图片高度
        String filename=ServletUtilities.saveChartAsPNG(chart, 800, 600,
info,null);
        //将图表的热点信息在 HTML 页面中输出，pw 代表页面的输出流，map 是定义热点的 Map 标签 ID
        //info 参数就是图片的热点信息
        ChartUtilities.writeImageMap(pwPrintWriter, "map", info, false);
%>
```

(2) 在项目的 web.xml 文件中配置 DisplayChart Servlet。前面已经讲过，这里不再详述。

(3) 在使用 DisplayChart 显示 Web 服务器临时目录下的统计图片时，需要传入一个 filename 的参数。因此，如果需要使用 DisplayChart 来显示临时目录下的统计图片，则应该指定一个 filename 参数，即在第一步中创建的生成带热点交互功能的统计图表 JSP 页面中加入如下代码。

```
<img src="DisplayChart?filename=<%=filename%>"  width="720" height="450"
usemap="#map">
```

(4) 从上面的代码中可以看出，该统计图指定的 URLGenerator 对应的 URL 为 piechart.jsp 页面。这个页面将读取该统计图发送的请求参数，并根据不同链接显示不同的统计图。piechart.jsp 页面的 Java 脚本代码如下。

```
<%@ page contentType="text/html;charset=GBK"%>
<%@ page import="org.jfree.chart.ChartFactory, org.jfree.chart.JFreeChart,
org.jfree.chart.plot.PlotOrientation" %>
<%@ page
import="org.jfree.chart.servlet.ServletUtilities,org.jfree.data.category.*"
%>
<%@ page import="org.jfree.chart.title.*"%>
<%@ page import="org.jfree.chart.plot.*"%>
<%@ page import="org.jfree.chart.axis.*"%>
<%@ page import="java.awt.*"%>
<%
            CategoryDataset dataset;
            //取得地址栏里的查询字符串
            String queryString = request.getQueryString();
            //将地址栏里的查询字符串解码成正确的编码形式
            queryString = java.net.URLDecoder.decode(queryString, "UTF-8");
            System.out.println(queryString);
            String category = null;
            //取出查询字符串的请求参数
            for (String param : queryString.split("&")) {
                String[] nameValue = param.split("=");
                category = nameValue[0].equals("category") ? nameValue[1]
                        : category;
            }
            String subTitle = "";
            //根据传过来的不同的请求参数决定 dataset 值，以及 subTitle 的字符串值
            if (category.equals("联想笔记本电脑 G450") || category.equals("戴尔
笔记本")) {
```

```
                dataset = leeGetDataSet();
                subTitle = "电脑";
            } else {
                dataset = getDataSet();
                subTitle = "手机";
            }
            //定义统计图的标题
            String title = subTitle + "机电销量统计图";
            //生成一个统计图实例
            JFreeChart chart = ChartFactory.createBarChart3D(title, "月份", "
销量",
                    dataset, PlotOrientation.VERTICAL, true, false, false);
            //重新设置图标标题，改变字体
            chart.setTitle(new TextTitle("机电销量统计图", new Font("黑体",
Font.ITALIC,
                    22)));
            //取得统计图标的第一个图例
            LegendTitle legend = chart.getLegend(0);
            //修改图例的字体
            legend.setItemFont(new Font("宋体", Font.BOLD, 14));
            CategoryPlot plot = (CategoryPlot) chart.getPlot();
            //取得横轴
            CategoryAxis categoryAxis = plot.getDomainAxis();
            //设置横轴显示标签的字体
            categoryAxis.setLabelFont(new Font("宋体", Font.BOLD, 22));
            //分类标签以 45 度角倾斜
            categoryAxis

    .setCategoryLabelPositions(CategoryLabelPositions.UP_45);
            categoryAxis.setTickLabelFont(new Font("宋体", Font.BOLD, 18));
            //取得纵轴
            NumberAxis numberAxis = (NumberAxis) plot.getRangeAxis();
            //设置纵轴显示标签的字体
            numberAxis.setLabelFont(new Font("宋体", Font.BOLD, 22));
            String filename = ServletUtilities.saveChartAsPNG(chart, 650, 390,
                    null, session);
        %>
    <%!//单击电脑类的产品销售量时，要生成的统计图的数据源
    private static CategoryDataset leeGetDataSet() {
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        dataset.addValue(3000, "联想笔记本电脑 G450", "09 年 10 月");
        dataset.addValue(2800, "联想笔记本电脑 G450", "09 年 11 月");
        dataset.addValue(2100, "联想笔记本电脑 G450", "09 年 12 月");
        dataset.addValue(3200, "联想笔记本电脑 G450", "10 年 01 月");
        dataset.addValue(2800, "惠普电脑", "10 年 01 月");
        dataset.addValue(2680, "惠普电脑", "10 年 02 月");
        dataset.addValue(2690, "联想笔记本电脑 G450", "10 年 02 月");
        dataset.addValue(1830, "戴尔笔记本", "10 年 02 月");
        dataset.addValue(3490, "惠普电脑", "10 年 03 月");
        dataset.addValue(1890, "联想笔记本电脑 G450", "10 年 03 月");
        dataset.addValue(2640, "戴尔笔记本", "10 年 03 月");
```

```
        dataset.addValue(3180, "惠普电脑", "10年04月");
        return dataset;
    }
    //单击手机类的产品时，要生成的统计图的数据源
    private static CategoryDataset getDataSet() {
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        dataset.addValue(3500, "诺基亚手机", "09年10月");
        dataset.addValue(4200, "索尼手机", "09年10月");
        dataset.addValue(3800, "诺基亚手机", "09年11月");
        dataset.addValue(2300, "索尼手机", "09年11月");
        dataset.addValue(2590, "诺基亚手机", "09年12月");
        dataset.addValue(1590, "索尼手机", "09年12月");
        dataset.addValue(3180, "诺基亚手机", "10年01月");
        dataset.addValue(1200, "索尼手机", "10年01月");
        dataset.addValue(3140, "诺基亚手机", "10年02月");
        dataset.addValue(940, "索尼手机", "10年02月");
        return dataset;
    }%>
```

(5) 从上面的 piechart.jsp 页面代码中可以看出：本页面与前面一个页面类似的是，本页面一样使用了 ServletUtilities 将统计图导出到 Web 服务器的临时目录下，因此在 JSP 页面中也需要使用 DisplayChart 来显示该统计图。下面是 JSP 页面中显示该统计图的HTML 标签代码。

```
<img src="DisplayChart?filename=<%=filename%>" width="650" height="390"
border="0" />
```

12.5.4 运行结果

在浏览器中浏览上面创建的第一个页面，即在网页中生成带交互功能的机电销量统计图页面，将看到如图 12-7 所示的页面。

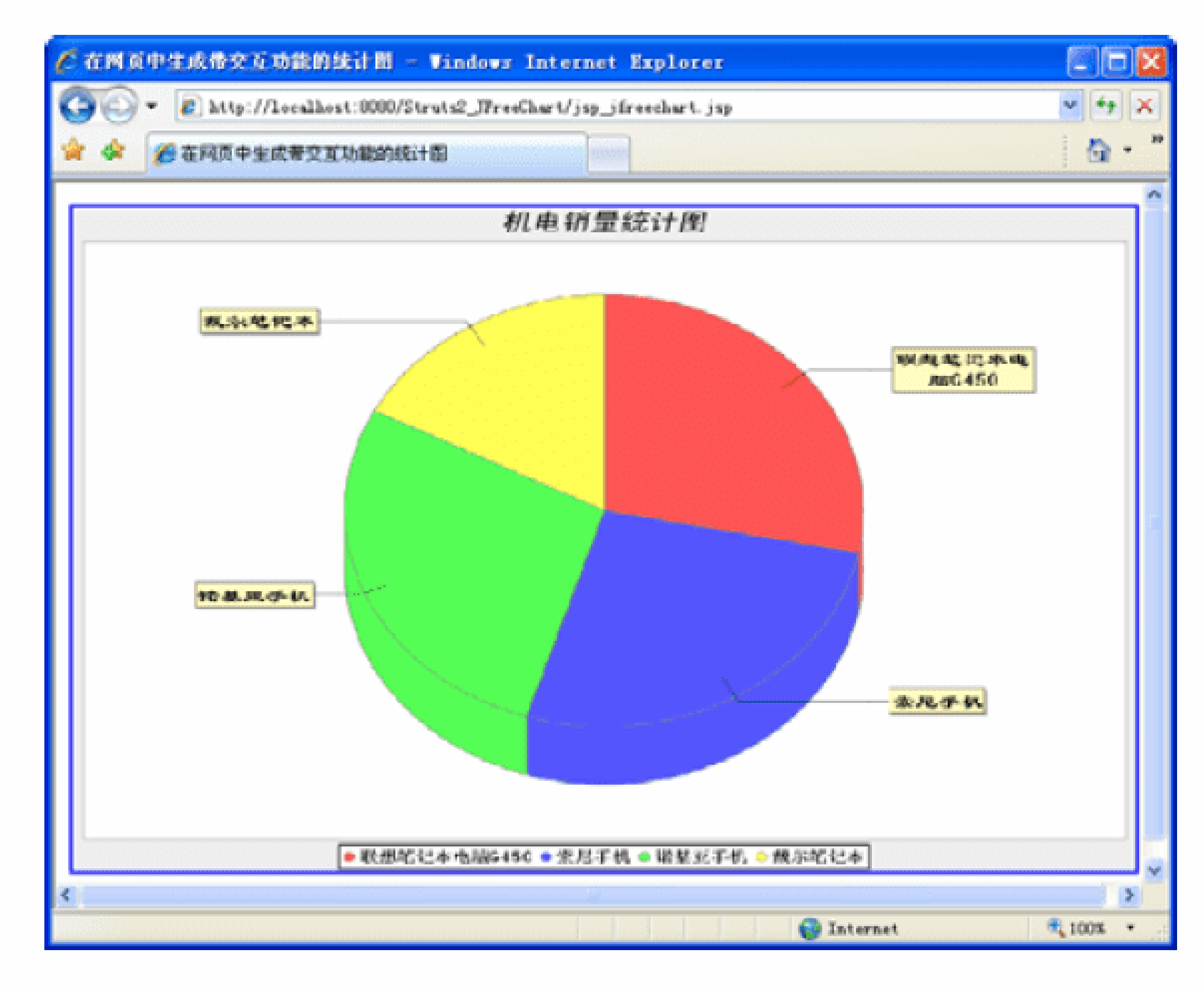


图 12-7 生成带交互功能的统计图

如果我们在上图 12-7 所示的页面中单击“联想笔记本 G450”和“戴尔笔记本”中任意一个部分，即可看到如图 12-8 所示的柱状统计图。

如果单击上图 12-7 所示的页面中的“诺基亚手机”和“索尼手机”的任意一个部分时，即可看到如图 12-9 所示的柱状统计图。

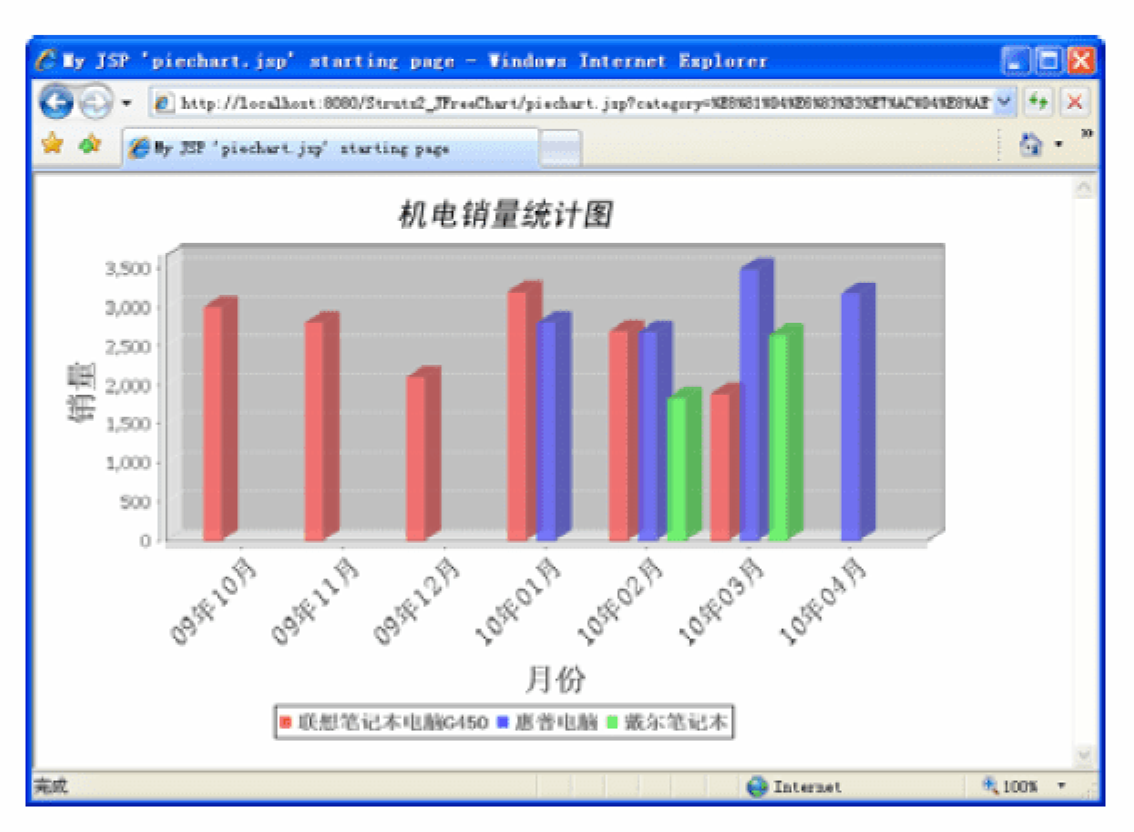
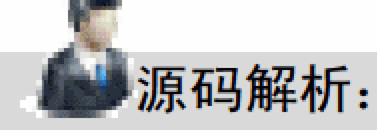


图 12-8　使用热点交互功能生成柱状图　　　图 12-9　使用热点交互功能生成柱状图

12.5.5　实例分析



源码解析：

在上案例中，对于 piechart.jsp 页面而言，它是完全自由的，可以非常灵活地输出任意内容，既可以再次生成统计图表来显示处理结果，也可以使用文本内容来输出处理结果。通过这种方式，可以实现非常灵活的用户交互功能。

12.6　在Struts 2 应用中使用JFreeChart

将 JFreeChart 与 Struts 2 整合后，对 JFreeChart 统计图表的开发有一定的简化作用。本节将为读者介绍如何在 Struts 2 中使用 JFreeChart 统计图表。



视频教学：光盘/videos/12/JFreeChartExample.avi　　　　长度：6 分钟

12.6.1　基础知识——在Struts 2 应用中使用JFreeChart

Struts 2 是一个 Java EE 应用的 Web 层解决方案，将底层数据以各种形式显示出来也是其主要任务。结合 JFreeChart 项目后，Struts 2 可以直接以 JFreeChart 图表作为表现层组件，直接使用 JFreeChart 图表来显示 Action 的处理结果。

1．安装JFreeChart插件

为了在 Struts 2 中使用 JFreeChart 统计图表，Struts 2 提供了 JFreeChart 插件支持。借助于 JFreeChart 插件的支持，Struts 2 可以非常方便地使用 JFreeChart 统计图表。

JFreeChart 插件的主要作用就是在页面中显示 Action 中的 JFreeChart 对象。实际上，JFreeChart 插件所完成的工作非常有限。即使在 Struts 2 中整合了 JFreeChart 框架，依然需要手动创建 JFreeChart 实例，这是非常繁琐的事情。最理想的状况就是只提供需要显示的数据，而 Struts 2 对 JFreeChart 进行封装，自动生成 JFreeChart 图表，并将其显示在 HTML 页面上。



Struts 2 对 JFreeChart 的封装非常有限，即使使用 Struts 2 整合 JFreeChart，依然需要在 Action 中手动创建 JFreeChart 实例，这点让人感到非常麻烦。

与 Struts 2 的其他插件类似，安装 JFreeChart 插件非常容易，只需将 Struts 2 的 JFreeChart 插件复制到 Web 应用的 WEB-INF/lib 路径下即可。

在研究 Struts 2 的 JFreeChart 时，发现 JFreeChart 插件与 JasperReports 插件有一个相同的问题：查看 struts2-jfreechart-plugin-2.1.8.1.jar 文件中的 struts-plugin.xml 文件时，发现该配置文件中代码如下。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="jfreechart-default" extends="struts-default">
        <result-types>
            <result-type name="chart"
class="org.apache.struts2.dispatcher.ChartResult">
                <param name="height">150</param>
                <param name="width">200</param>
            </result-type>
        </result-types>
    </package>
</struts>
```

从该文件中看出，JFreeChart 插件的主要作用就是定义了一个名为 chart 的 Result 类型。JFreeChart 插件定义的包是从 struts-default 扩展而来，这样就可以以简单的方式在 Struts 2 应用中使用 JFreeChart 统计图表了。

除了要将上面的 struts2-jfreechart-plugin-2.1.8.1.jar 文件复制到 Web 应用的 WEB-INF/lib 路径下之外。还需要将 JFreeChart 的二进制类库文件复制到 Web 应用的 WEB-INF/lib 路径下。

经过上面的步骤，即完成了 Struts 2 和 JFreeChart 的整合。

2．JFreeChart在Struts 2 中的使用

正如前面介绍的，使用 Struts 2 整合 JFreeChart 后，对开发 JFreeChart 并没有提供太多的简便，依然需要手动创建 JFreeChart 实例。

1)　创建 Action

如果需要的 Action 中的统计图表能被 Struts 2 处理，则要求该 JFreeChart 类型的属性名为 chart。在 Struts 2 的 Action 中创建生成统计图表的 JFreeChart 实例，需要在 Action 中定义一个返回 JFreeChart 对象的方法，即代码如下。

```
public class JFreeChartAction extends ActionSupport {
    //用于输出统计图表的属性必须是 chart
    private JFreeChart chart;
    //返回 JFreeChart 统计图表的 getter 方法
    public JFreeChart getChart() {
    /*生成统计图表的代码，和前面使用 JFreeChart 生成的统计图表代码并无二样*/
    }
}
```

上面 Action 类继承 ActionSupport 类，因此无需提供 execute()方法，该方法返回一个 success 字符串作为逻辑视图，该 Action 直接使用 ActionSupport 的 execute()方法作为系统的处理逻辑。从上面的代码中不难发现，上面的 Action 有如下两个非常不灵活的地方。

- 需要手动创建 JFreeChart 实例，与创建普通 JFreeChart 图表没有太大的区别。
- Action 的统计图表属性名必须是 chart，这种硬编码方式提供的属性大大降低了 Action 的灵活性。



JFreeChart 插件要求用户的 Action 中 JFreeChart 类型的属性名只能是 chart，这种硬编码方式提供的统计图表大大降低了 Action 的灵活性。

2)　配置 Action

配置 JFreeChart 的 Action 非常简单，只需要为该 Action 指定一个类型为 chart 的 Result。该 Result 将使用 JFreeChart 统计图表来作为视图组件。配置类型为 chart 的 Result 时，可以指定两个参数：width 和 height，这两个参数分别指定统计图的宽和高。

因为前面已经定义了名为 jfreechart-default 的包，该包扩展了 Struts 2 系统的 struts-default 包，在 struts-default 包里增加了一个名为 chart 的 Result 类型，因此应该在 Struts 2 应用中自定义包继承 jfreechart-default 包即可。如下配置所示。

```
<!-- 配置包，继承 jfreechart-default 包 -->
    <package name="jfreechart" namespace="/jfreechart"
extends="jfreechart-default">
        <!-- 定义一个名为 jfreeChart 的 Action-->
        <action name="jfreeChart"
class="com.struts2.actions.JFreeChartAction">
            <result type="chart">
                <!-- 定义 JFreeChart 报表的大小 -->
                <param name="width">800</param>
                <param name="height">600</param>
            </result>
        </action>
    </package>
```

上面配置文件配置了一个名为 jfreeChart 的 Action。该 Action 将产生一个类型为 chart 的 Result。jfreeChart 的 Action，即可生成统计图。

12.6.2 实例描述

为了给读者讲清楚 JFreeChart 在 Struts 2 中的应用，我把前面为朋友公司做的销售情况统计图改编成使用 Struts 2 与 JFreeChart 整合的应用。这样，在他们公司的后台管理系统中就可以查看这个应用了，而不需要生成一张图片。

12.6.3 实例应用

【例 12-6】 在 Struts 2 应用中使用 JFreeChart 生成“机电销售统计图”。

(1) 将 struts2-jfreechart-plugin-2.1.8.1.jar 文件和 Struts 2 相关的六个文件一同引入程序的 WEB-INF/lib 下。

(2) 在 com.struts2.actions 包下新建 JFreeChartAction 类，定义统计图表属性名为 chart，并编写生成统计图的方法。JFreeChartAction 类内容如下。

```
package com.struts2.actions;
import java.awt.Font;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PiePlot3D;
import org.jfree.chart.title.LegendTitle;
import org.jfree.chart.title.TextTitle;
import org.jfree.data.general.DefaultPieDataset;
import com.opensymphony.xwork2.ActionSupport;
public class JFreeChartAction extends ActionSupport {
    //用于输出统计图表的属性必须是 chart
    private JFreeChart chart;
    //返回 JFreeChart 统计图表的 getter 方法
    public JFreeChart getChart() {
        chart=ChartFactory.createPieChart3D(
                "机电销量统计图", //图表标题
                getDataset(), //数据
                true, //是否显示图例
                false,//是否显示工具提示
                false//是否生成 URL
                );
        //重新设置图表标题，改变字体
        chart.setTitle(new TextTitle("机电销量统计图",new Font("黑体
",Font.ITALIC,22)));
        //取得统计图表的第一个图例
        LegendTitle legendTitle=chart.getLegend(0);
        //改变图例的字体
        legendTitle.setItemFont(new Font("宋体",Font.BOLD,14));
        //获得饼图的 Plot 对象
```

```
        PiePlot3D plot3D=(PiePlot3D)chart.getPlot();
        //设置饼图各部分的标签字体
        plot3D.setLabelFont(new Font("隶书",Font.BOLD,18));
        //设定背景透明度(0-1.0 之间)
        plot3D.setBackgroundAlpha(0.9f);
        //设定前景透明度(0-1.0 之间)
        plot3D.setForegroundAlpha(0.5f);
        return chart;
    }
    //返回饼图的底层 Dataset 的工具方法
    private DefaultPieDataset getDataset(){
        DefaultPieDataset dataset=new DefaultPieDataset();
        dataset.setValue("联想笔记本电脑 G450",2000);
        dataset.setValue("索尼手机",1900);
        dataset.setValue("诺基亚手机",1980);
        dataset.setValue("戴尔笔记本",1230);
        return dataset;

    }
}
```

(3) 在 Struts 2 配置文件(struts.xml)中配置该 Action，需要配置类型为 char 的 Result。struts.xml 文件内容如下。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- 指定拦截器的 DTD 信息 -->
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <!-- 通过常量配置 Struts2 所使用的解码集 -->
    <constant name="struts.i18n.encoding" value="gbk" />
    <constant name="struts.devMode" value="true" />
    <!-- 配置包，继承 jfreechart-default 包 -->
    <package name="jfreechart" namespace="/jfreechart"
extends="jfreechart-default">
        <!-- 定义一个名为 jfreeChart 的 Action-->
        <action name="jfreeChart"
class="com.struts2.actions.JFreeChartAction">
            <result type="chart">
                <!-- 定义 JFreeChart 报表的大小 -->
                <param name="width">800</param>
                <param name="height">600</param>
            </result>
        </action>
    </package>
</struts>
```

(4) 在 WEB-INF 目录下的 web.xml 文件中配置 Struts 2 控制器——FilterDispatcher 类。代码如下。

```
<filter>
    <filter-name>struts2</filter-name>

    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-cl
ass>
</filter>
<filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>*.action</url-pattern>
</filter-mapping>
```

(5) 在 WebRoot 目录下新建 struts.jsp 页面，在页面中请求 JFreeChartAction 类，即如下所示。

```
<img src="jfreechart/jfreeChart.action"  border="0">
```

12.6.4 运行结果

运行 struts.jsp 页面，出现如图 12-10 所示的页面。

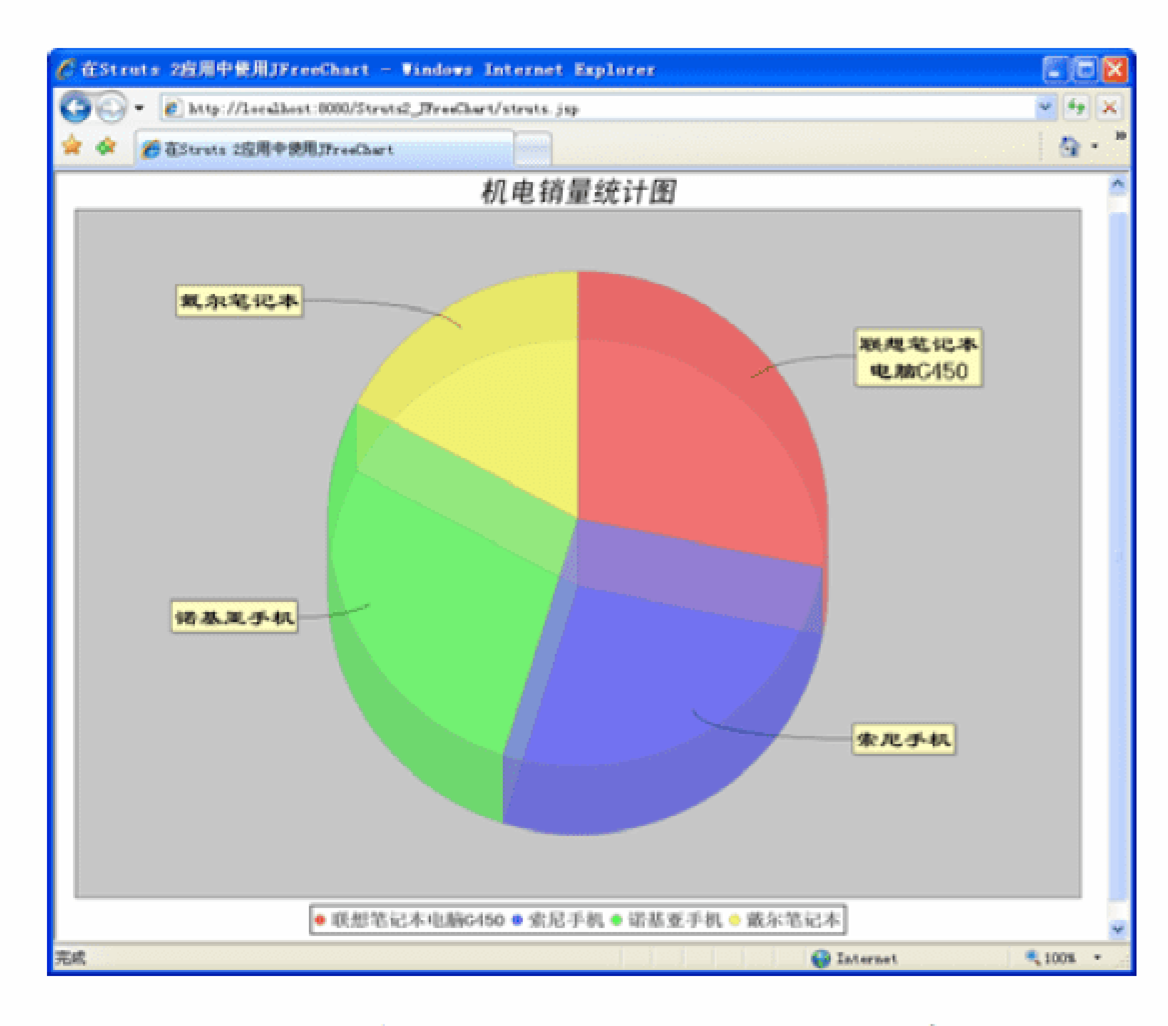
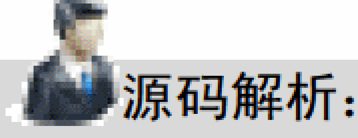


图 12-10 整合Struts 2 和JFreeChart生成统计图

12.6.5 实例分析



源码解析：

在上案例中，在页面中以<img src="jfreechart/jfreeChart.action" border="0">方式请求 JFreeChartAction 类，JFreeChartAction 类继承了 ActionSupport 类，无需重写 execute()方法，当请求 JFreeChartAction 类时，系统执行父类中的 execute()方法，并返回 success 字符串作为逻辑视图。在配置 Action 类时配置了一个 chart 类型的 Result，系统会自动加载 JFreeChartAction 类中的 chart 对象，生成统计图表。

12.7 常见问题解答

12.7.1 JFreeChart中文乱码问题



JFreeChart 中文乱码问题！
网络课堂：http://bbs.itzcn.com/thread-11013-1-1.html

今天我在做一个由 JFreeChart 生成的柱状图，它的标题是中文的，可是在页面上显示的都是“框框”，让我郁闷了半天，怎么解决 JFreeChart 的中文乱码问题？

【解决办法】：使用 JFreeChart 生成统计图，出现乱码问题，是属性设置问题，用以下方式可以解决，请看下面的代码。

```
    chart.getLegend().setItemFont(CHART FONT);
    chart.getTitle().setFont(CHART FONT);
    // 获得坐标轴对象 Axis,横轴对象:
    Axis axis=chart.getCategoryPlot().getDomainAxis();
    // 纵轴对象:
    Axis axis1=chart.getCategoryPlot().getRangeAxis();
    axis.setLabelFont(CHART_FONT); // 坐标轴的字体,下同
    axis.setTickLabelFont(CHART_FONT);  //图中间的字体,
    axis1.setLabelFont(CHART_FONT);
```

不管是柱状图、折线图还是饼图，这段代码都有效。

12.7.2 在unix操作系统下使用JFreeChart问题



在 unix 操作系统下使用 JFreeChart 问题！
网络课堂：http://bbs.itzcn.com/thread-11014-1-1.html

我电脑的操作系统是 unix 操作系统，使用 JFreeChart 生成统计图时遇到下面问题。

```
Can't connect to X11  window
```

【解决办法】：遇到此问题，可以通过在启动时加-Djava.awt.headless=true 参数来解决。

12.7.3 使用JFreeChart生成统计图出现UnsatisfiedLinkError错误



使用 JFreeChart 生成统计图出现 UnsatisfiedLinkError 错误！
网络课堂：http://bbs.itzcn.com/thread-11015-1-1.html

今天在使用 JFreeChart 与 Struts 2 整合生成统计图时，出现 java.lang.UnsatisfiedLinkError 的错误，这是怎么回事？我查了好多资料，没弄明白。急！

【解决办法】：遇到 java.lang.UnsatisfiedLinkError 的错误很可能是 JDK 安装、系统配置或者没装 JDK 补丁问题。在 HP-UX B.11.23 机器上用 JDK1.4.2_08 就出现了下面错误。

```
java.lang.UnsatisfiedLinkError:initIDs at java.awt.Font.initIDs(Native
Method)
```

把安装的 JDK1.4.2_08 换成 JDK1.4.2_02 就可以了。

12.7.4 每次生成JFreeChart统计图都会抛出异常



每次生成 JFreeChart 统计图都会抛出异常！
网络课堂：http://bbs.itzcn.com/thread-11018-1-1.html

每次生成 JFreeChart 统计图的时候，都会抛出异常，异常指出问题出在 ChartFactory.createXYLineChart(picName,"时间轴","数据",xydataset,true,true,false)这行代码上(或 create 其他类型的 chart 代码上)。这是怎么回事？

【解决办法】：由于 JFreeChart 用到的画图库是 Java AWT，所以需要确保 JVM 运行在 headless 模式下，如果在 Unix 系统中使用 JFreeChart，需要在 Tomcat 的 bin 目录下 catalina.sh 文件中 run 和 start 两处添加下面代码。

```
java.awt.headless=true
```

12.7.5 JFreeChart生成的统计图时间轴中时间的显示格式问题



JFreeChart 生成的统计图时间轴中时间的显示格式问题！
网络课堂：http://bbs.itzcn.com/thread-11020-1-1.html

【解决办法】：在生成两组或者多组数据的 chart 时，分为以下两种情况。

一种是 X 轴和 Y 轴数据都仅仅只是数据，则可以直接使用 ChartFactory.createXYLineChart(picName,"X 轴数据","Y 轴数据",xydataset,true,true,false)，xydataset 是通过 XYSeriesCollection 收集 XYSeries 获取的数据生成，代码如下。

```
XYSeries[] xyseries=new XYSeries[count];
for(int i = 0; i < count; i ++){
    xyseries[i] = new XYSeries(name[i]);
}
xyseries[1].add(double arg0,double arg1);
xyseries[2].add(double arg0,double arg1);
```

两组数据都必须以双精度格式传入，在 chart 统计图中的 X 轴和 Y 轴数据也都会以双精度格式显示。

另一种是 X 轴数据是时间，Y 轴数据为与这个时间对应的一个有一定精确度的数据，这种情况就得使用 ChartFactory.createTimeSeriesChart(picName,"时间轴","数据",xydataset,true,true,false)，xydataset 是通过 TimeSeriesCollection 收集 TimeSeries 获取的数据生成，代码如下。

```
TimeSeries timeseries[] = new TimeSeries[count];
for(int i = 0; i < count; i ++){
    timeseries[i] = new TimeSeries(name[i],Minute.class);
}
timeseries[0].addOrUpdate(new Minute(minute,hour,date,month,year), double
arg1);
timeseries[1].addOrUpdate(new Minute(minute,hour,date,month,year), double
arg1);
```



这里使用 addOrUpdate 而不使用 add 的原因是：如果使用 add 的话，传入一次数据，生成一个 chart 统计图，当下一次其他某个地方再调用这个方法生成 chart 统计图的时候，就会报出数据冲突的异常信息，估计是上次传入的数据没有清除掉的原因，所以为了防止这种情况一般都使用 addOrUpdate。

两组数据是以(时间、双精度数据)成对传入的，X 轴将会是一个时间轴，而且会以通用的时间格式进行显示，显示的时间格式可以在代码中定制，详细代码如下。

```
DateAxis dateaxis = (DateAxis)xyplot.getDomainAxis();
dateaxis.setDateFormatOverride(new SimpleDateFormat("时间格式"));
```

12.8　习　　题

一、填空题

(1) 在整合 Struts 2 与 JFreeChart 时，需要配置 struts.xml 文件，代码如下。

```
<struts>
    <package name="jfreechart-default" extends="struts-default">
        <result-types>
            <result-type name="__________"
class="org.apache.struts2.dispatcher.ChartResult">
                <param name="height">150</param>
                <param name="width">200</param>
            </result-type>
        </result-types>
    </package>
</struts>
```

划横线处应该填__________。

(2) 柱状图的 DataSet 一般是用 CatagoryDataset 接口，具体实现类是__________。

(3) 生成时间顺序图需要使用 XYDataset 实例作为统计图的底层数据，具体实现类是__________。

二、选择题

(1) JFreeChart 开发步骤可以分为 4 大步，顺序是__________。

a. 提供一个 Dateset 实例，该实例里包含了创创建统计图表的数据。

b. 得到了 JFreeChart 对象后，可以调用 setTitle 来修改统计图表的标题，或者调用 getLegend() 方法来获得指定索引的图表图例，取得图例对象后即可修改图表的图例。

c. 使用 ChartFactory 的多个工厂方法 createXxxChart()来创建统计图表，统计图表就是一个 JFreeChart 对象。

d. 通过 JFreeChart 对象的 getPlot()方法，即可获得图表的 Plot 对象，该对象对应于统计图表的实际图表部分，可以调用 Plot 对象的方法来修改图表中的各种显示内容。

A. a、b、c、d　　B. a、b、d、c

C. a、c、b、d　　D. b、a、c、d

(2) 生成带交互功能的统计图需要使用 JFreeChart 项目的 3 个核心 API，分别是__________。

A. XxxToolTipGenerator XxxURLGenerator ChartRendererInfo

B. XxxToolTipGenerator XxxDataset ChartFactory

C. JFreeChart XxxURLGenerator ChartRendererInfo

D. JFreeChart XxxDataset ChartFactory

(3) 配置类型为 chart 的 Result 时，可以指定两个参数：__________和__________，这两个参数分别指定统计图的宽和高。

A. width　tall　　B. wide　tall　　C. width　height　　D. wide　height

三、上机练习

上机练习：Struts 2 与 JFreeChart 整合生成“企业备案图”。

要求：使用 JFreeChart 与 Struts 2 整合，在 JSP 页面中生成一张“企业备案图”。这张“企业备案图”横轴上显示的是数据，纵轴上显示的是各局名称，如图 12-11 所示。

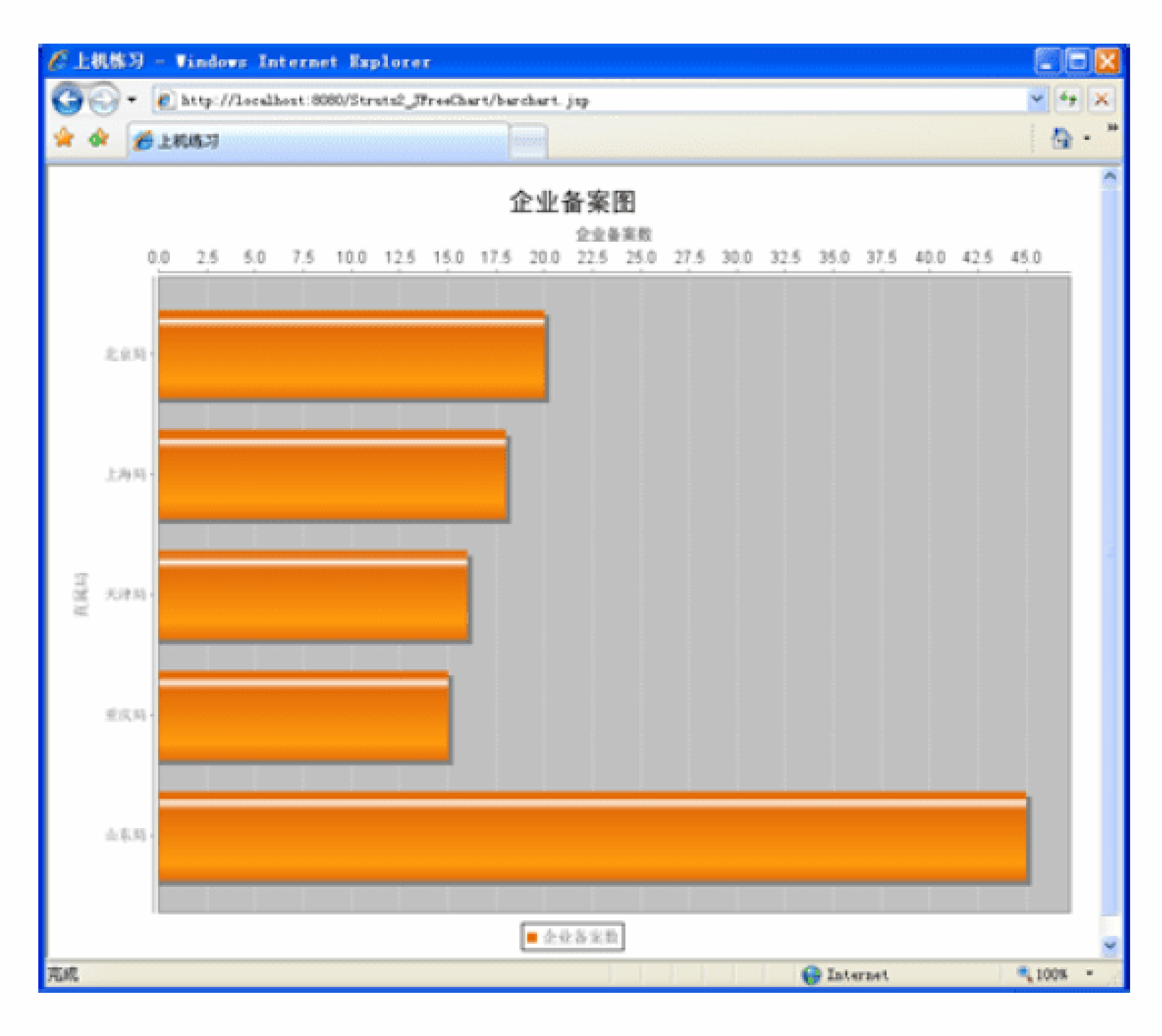


图 12-11　JFreeChart与Struts 2 整合生成“企业备案图”



第 13 章　当Struts 2 碰见Ajax

内容摘要：

Ajax 非常完美地改善了用户体验，使用户体验了一种连续的状态，避免了传统的 Web 应用进行请求——等待——响应的流程。Ajax 使用户可以连续发送多次异步请求，不需要等待服务器响应。当服务器的响应成功返回浏览器时，浏览器利用 DOM 将服务器响应数据加载到当前页面的相应容器中。

Struts 2 提供了完备的 MVC 功能，同时也提供了简单易用的 Ajax 支持。Struts 2 的 Ajax 支持需要建立在 DWR 和 Dojo 这两个非常成熟的 Ajax 框架。Struts 2 在这两个框架的基础上，进行了进一步的封装，从而简化了 Ajax 的开发。

本章主要讲解 Struts 2 的 Ajax 支持。采用 Ajax 方式输入校验，如果输入不合法的数据，系统自动显示校验提示。还允许使用 Dojo 异步获取数据请求，并提供了一系列的 Ajax 标签来简化 Ajax 开发。

学习目标：

- 掌握 Ajax 的输入校验。
- 了解 DWR 框架的使用。
- 掌握 JSON 串作为数据的载体。
- 熟悉 Dojo 框架的使用。
- 掌握 Struts 2 的 Ajax 标签。

13.1 用户注册校验

Ajax 的输入校验是服务器端校验，不是客户端校验，这种服务器端校验是以异步方式进行的，不需要浏览者显示提交校验请求，当浏览者输入完成后，系统自动完成校验。

视频教学：光盘/videos/13/InputAjax.avi　　　　长度：7 分钟

13.1.1 基础知识——基于Ajax的输入校验

Struts 2 的 Ajax 校验建立在 DWR 和 Dojo 两个框架之上，其中 DWR 负责实现在 JavaScript 中调用远程 Java 方法，Dojo 则负责实现页面上的效果。本节先让读者了解一下 Ajax 和 DWR 的相关简介，然后学习 DWR 的下载与配置应用，最后结合实例深入学习 Ajax 的输入校验。

1. Ajax简介

Ajax 不是一种新的编程语言，而是一种用于创建更好更快以及交互性更强的 Web 应用程序的技术。

通过 Ajax，可以使用 JavaScript 的 XMLHttpRequest 对象来直接与服务器进行通信。通过这个对象，编写的 JavaScript 可以在不重载页面的情况与 Web 服务器交换数据。

1) Ajax 的优点

首先介绍 Ajax 的主要优点。

(1) 有针对性的数据传输。

Ajax 的原则是“按需取数据”，可以最大限度的减少冗余请求，和响应对服务器造成的负担。例如，新用户名的检测，如果使用 Ajax 技术进行检测，它可以只将用户名信息传送到服务器，而不需要传送整个表单数据。从服务器端获取相关返回数据后，它又可以只向客户端传输这些反馈数据，而不需要刷新整个页面而导致页面中其他代码内容被重新传输。

(2) 无刷新更新页面。

减少用户心理和实际的等待时间。当要读取大量的数据的时候，不用像 Reload 那样出现白屏的情况，Ajax 使用 XMLHTTP 对象发送请求并得到服务器响应，在不重新载入整个页面的情况下用 JavaScript 操作 DOM 最终更新页面。所以在读取数据的过程中，用户所面对的不是白屏，是原来的页面内容(也可以一个 Loading 的提示框让用户知道处于读取数据过程)，只有当数据接收完毕之后才更新相应部分的内容。这种更新是瞬间的，用户几乎感觉不到。

(3) 基于公开的标准。

Ajax 技术是基于已经被各大浏览器和平台都支持的公开标准的技术。组成 Ajax 技术的大多数技术都经过很多年的实践考验。

(4)　跨平台跨浏览器的兼容性。

占据市场份额最大的两个浏览器是 IE 和基于 Mozilla 的 FireFox，而它们都支持在浏览器上轻松创建基于 Ajax 的 Web 应用。这也是 Ajax 应用变得如此流行的一个最重要的原因。

(5)　技术独立性。

和 Ajax 技术浏览器的独立性相同，该技术也兼容所有标准型的服务器和服务端语言，如 PHP、ASP、ASP.Net、Perl、JSP 和 Cold Fusion 等。这使得 Ajax 开发具有独立性，所有的开发人员都能使用并且一起讨论相同的表现层。

2)　Ajax 的缺点

没有绝对完美的事物，Ajax 也有它的一些缺点。

(1)　浏览器的通用性，每个客户的浏览器不尽相同、版本也不一致，有可能会造成一些操作无法进行。

(2)　客户端会过肥，太多程序代码在客户端也会造成开发上的成本。

(3)　可能会暴露服务端，有可能被恶意攻击、窜改，而造成安全上的漏洞。

3)　Ajax 的工作原理

Ajax 通过 Ajax 引擎(其核心是 XMLHttpRequest 对象)与服务器进行交互。Ajax 引擎向服务器发送请求后，Ajax 在服务器状态发生变化时通知 JavaScript 脚本程序来处理相应的件。Ajax 的工作原理如图 13-1 所示。

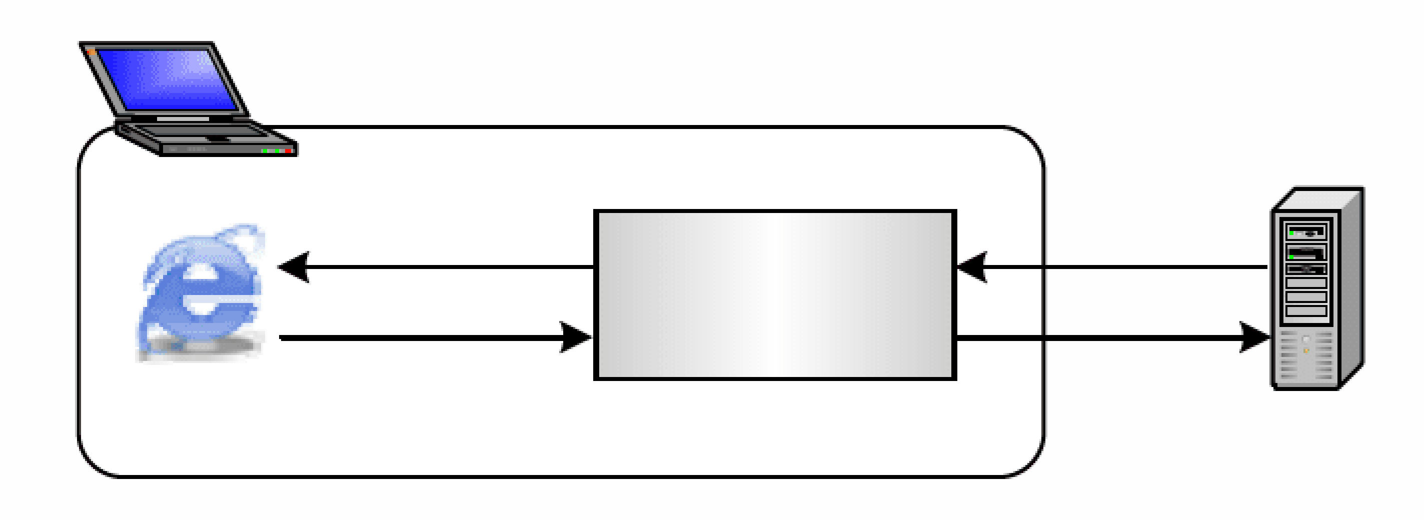


图 13-1　Ajax的工作原理

2. DWR的简介

DWR(Direct Web Remoting)是一个用于改善 Web 页面与 Java 类交互的远程服务器端 Ajax 开源框架，可以帮助开发人员开发包含 Ajax 技术的网站。它可以允许在浏览器里的代码使用运行在 Web 服务器上的 Java 函数，好像它就在浏览器里一样。

利用一个 Ajax 框架(指 DWR)构造一个应用程序，它直接从浏览器与后端服务进行通信。如果使用得当，这种强大的力量可以使应用程序更加自然和响应灵敏，从而提升用户的浏览体验。

Struts 2 中的 ajax 主题是在 XHTML 主题基础上的扩展，增加了 Ajax 数据校验功能。Ajax 输入校验不是基于客户端，而是基于服务器端，是以异步方式实现的校验方式。



目前，Struts 2 中的 Ajax 输入验证使用的是 DWR 框架。

3．下载与配置DWR

使用 DWR 框架需要在 Web 应用中加载 DWR 的 JAR 文件 dwr.jar，可以进入到 DWR 的官方网站“http://directwebremoting.org/dwr/downloads/index.html”进行下载，下载完成后，将 dwr.jar 复制粘贴到 Web 应用程序的 WEB-INF/lib 目录下。



目前 DWR 的最新版本是 3.0，但是 Struts 2 的 2.1.8 版本不支持最新版本，因此只能使用 DWR 的 2.0 或以下版本，本书中使用的是 DWR 1.1.3 版本，相对比较稳定。

仅下载 DWR 的 Jar 包是不能在项目中使用 DWR 框架的，还需要在项目的 web.xml 文件中配置 DWR 的核心 Servlet，详细配置如下所示。

```
<servlet>
    <servlet-name>dwr</servlet-name>
    <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
    <init-param>
        <param-name>debug</param-name>
        <param-value>true</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>dwr</servlet-name>
    <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

增加了 DWR 的核心 Servlet 后，该 Servlet 负责将服务器端的 Java 方法暴露出来。具体被暴露的 Java 方法，在 dwr.xml 配置文件中指定。dwr.xml 文件需要放到 WEB-INF/lib 目录下，配置信息如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
    <!-- START SNIPPET: dwr -->
<!DOCTYPE dwr PUBLIC
    "-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
    "http://www.getahead.ltd.uk/dwr/dwr10.dtd">
<dwr>
    <!-- 定义所有需要被暴露的 Java 方法 -->
    <allow>
        <create creator="new" javascript="validator">
            <param name="class"
value="org.apache.struts2.validators.DWRValidator" />
        </create>
        <convert converter="bean"
            match="com.opensymphony.xwork2.ValidationAwareSupport" /><!-- 定
义一个转换器-->
    </allow>
    <signatures>
        <![CDATA[
        import java.util.Map;
        import org.apache.struts2.validators.DWRValidator;
```

```
        DWRValidator.doPost(String, String, Map<String, String>);
        ]]>
    </signatures>
</dwr>
    <!-- END SNIPPET: dwr -->
```

在 dwr.xml 配置文件中，将 org.apache.struts2.validators.DWRValidator 类创建成一个 JavaScript 对象，名为 validator。DWR 框架提供一种方式，允许在客户端调用 validator 的方法时，转换成调用 DWRValidator 实例的方法。



在具体的项目应用中，开发者只需利用 Struts 2 对 DWR 的封装即可。在 WEB-INF 路径下添加 dwr.xml 文件，而且 dwr.xml 文件的代码也是固定不变的。

13.1.2　实例描述

前几天在网上看到一件衣服很漂亮，我很喜欢于是就想买下来。首先注册一个账号，输入注册用户信息，当输入密码时，在输入框后面，显示了输入错误提示“用户密码必须为 6-20 位”，没办法只好换个符合要求的密码。这种情况估计经常上网的朋友都遇到过，其实这就是本节实例要实现的用户注册校验，检查用户输入信息是否合法，不合法给出相应提示信息。

13.1.3　实例应用

【例 13-1】 用户注册校验。

(1)　新建一个 Struts2_13 项目，为了让 DWR 的核心 Servlet 起作用，必须在 web.xml 文件中配置该核心 Servlet。配置 DWR 的核心 Servlet 的代码如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app 2 5.xsd">
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <!-- Struts 2 核心控制器配置-->
    <filter>
        <filter-name>struts</filter-name>

    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-
    class>
        <init-param>
            <param-name>actionPackages</param-name>
            <param-value>org.apache.struts2.showcase.person</param-value>
        </init-param>
    </filter>
```

```
    <filter-mapping>
        <filter-name>struts</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <!-- 配置 DWR 的核心 Servlet -->
    <servlet>
        <servlet-name>dwr</servlet-name>
        <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
        <init-param>
            <param-name>debug</param-name>
            <param-value>true</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>dwr</servlet-name>
        <url-pattern>/dwr/*</url-pattern>
    </servlet-mapping>
</web-app>
```

(2) 在上面 web.xml 文件中添加了 DWR 的核心 Servlet 后，该 Servlet 将负责暴露服务器端的 Java 方法，通过 dwr.xml 配置文件制定需暴露的 Java 方法，在项目 WEB-INF 目录下新建一个 dwr.xml 文件，在该文件中添加如下配置代码。

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- START SNIPPET: dwr -->
<!DOCTYPE dwr PUBLIC
    "-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
    "http://www.getahead.ltd.uk/dwr/dwr10.dtd">
<dwr>
    <allow>
        <create creator="new" javascript="validator">
            <param name="class"
value="org.apache.struts2.validators.DWRValidator" />
        </create>
        <convert converter="bean"
            match="com.opensymphony.xwork2.ValidationAwareSupport" />
    </allow>
    <signatures>
        <![CDATA[
        import java.util.Map;
        import org.apache.struts2.validators.DWRValidator;
        DWRValidator.doPost(String, String, Map<String, String>);
        ]]>
    </signatures>
</dwr>
<!-- END SNIPPET: dwr -->
```

(3) 在项目 src 目录下新建一个 com.gdupt.action 包，在该包下再新建一个 RegisterAction 类，用于处理用户注册请求，代码如下所示。

```
package com.gdupt.action;

import com.opensymphony.xwork2.ActionSupport;

public class RegisterAction extends ActionSupport {

    private String username;    //用户名
    private String password;    //密码
    private int age;            //年龄

    //下面是 username、password 和 age 属性的 get 和 set 方法，在此省略了。
    //……………………………………………..

    public String execute(){
        return SUCCESS;
    }
}
```

(4) 在项目 src 目录下新建一个 struts.xml 文件，在该文件中添加 RegisterAction 的配置，代码如下所示。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="default" extends="struts-default">
        <action name="userregister" class="com.gdupt.action.RegisterAction">
            <result name="input">/index.jsp</result>
            <result name="success">/success.jsp</result>
        </action>
    </package>
</struts>
```

(5) 在 RegisterAction 所在目录下，新建一个 RegisterAction-validation.xml 文件，在该文件中设置校验规则，来完成输入校验，本输入校验文件使用字段校验器风格来配置校验规则。配置代码如下所示。

```
<!DOCTYPE validators PUBLIC
        "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
        "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
    <field name="username">
        <field-validator type="requiredstring">
            <message>用户名是必填内容</message>
        </field-validator>
    </field>
    <field name="password">
        <field-validator type="requiredstring">
            <message>密码是必填的内容!</message>
        </field-validator>
    </field><field name="age">
```

```
        <field-validator type="int">
            <param name="min">1</param>
            <param name="max">130</param>
            <message>年龄必须在 1 到 130 之间</message>
        </field-validator>
    </field>
</validators>
```

(6) 新建一个 register.jsp 页面，为了实现 Ajax 校验，需要将表单设置成 Ajax 主题，并且设置 validate=“true”。当某个输入组件失去焦点时，系统会负责将输入内容发送到服务器端进行校验。该页面实现代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@taglib uri="/struts-tags" prefix="s"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <s:head theme="ajax"/>
    </head>
    <body>
        <s:form method="post" action="userregister" validate="true"
theme="ajax">
            <s:textfield label="用户名" name="username" />
            <s:password label="密码" name="password"></s:password>
            <s:textfield label="年龄" name="age"></s:textfield>
            <s:submit value="注册" />
        </s:form>
    </body>
</html>
```

(7) 新建一个 success.jsp 页面，当用户注册成功后跳转到 success.jsp 页面，代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>注册成功</title>
    </head>
    <body>
        恭喜您，注册成功！ <br>
    </body>
</html>
```

13.1.4 运行结果

打开 IE 浏览器，在地址栏中输入“http://localhost:8080/Struts2_13/register.jsp”，进入注册页面，如果在用户输入信息表单中输入不合法的信息或不输入任何信息，单击“注册”按钮提交表单，经过验证之后会在输入框上方给出相应的提示信息，执行效果如图 13-2 所示。



图 13-2　用户注册

13.1.5　实例分析


源码解析：

本实例实现 Struts 2 支持的 Ajax 校验，首先需要在 web.xml 文件中添加 DWR 的核心 Servlet 配置和在项目 WEB-INF 目录下添加 dwr.xml 文件。

然后新建一个 RegisterAction 类，在该 Action 中声明用户注册信息属性，如：username、password、age。接下来在 RegisterAction 类所在目录下，创建一个 RegisterAction-validation.xml 在该文件中设置校验规则，来完成输入校验。

最后创建一个用户注册页面 register.jsp，当用户输入不合法注册信息时，系统将自动提交校验，并给出提示信息。

13.2　JSON串传递顾客信息数据

XML 的作用就是利用规范的文档来格式化数据内容，所以在使用 XML 时，需要编写很多格式内容。从数据传输量上来看 JSON 显然要优于 XML，JSON 更轻量级一些，它没有像 XML 那样多的 Open 和 Closing 标记。同时在对数据的解析速度上，JSON 也要优于 XML。因此使用 JSON 传递数据相对比较方便。

视频教学：光盘/videos/13/json.avi　　长度：8 分钟
光盘/videos/13/json1.avi　　长度：5 分钟
光盘/videos/13/json2.avi　　长度：14 分钟

13.2.1　基础知识——使用JSON串作为数据的载体

JSON 插件是 Struts 2 的 Ajax 插件，使用 JSON 插件，可以使开发者非常灵活的开发 Ajax 应用，而且整个开发过程非常简单。本节将学习如何在 Struts 2 中使用 JSON 传递数据信息。

1. JSON概述

JSON(JavaScript Object Notation)是一种轻量级的数据交换格式。易于人阅读和编写，同时也易于机器解析和生成。它基于 JavaScript(Standard ECMA-262 3rd Edition - December 1999)的一个子集。JSON 采用完全独立于语言的文本格式，但是也使用了类似于 C 语言家族的习惯(包括 C、C++、C#、Java、JavaScript、Perl、Python 等)。这些特性使 JSON 成为理想的数据交换语言。

2. JSON构建结构

JSON 有两种构建结构。

(1) “名称/值”对的集合(A collection of name/value pairs)

不同的语言中，它被理解为对象(object)，记录(record)，结构(struct)，字典(dictionary)，哈希表(hash table)，有键列表(keyed list)，或者关联数组(associative array)。

(2) 值的有序列表(An ordered list of values)

在大部分语言中，它被理解为数组(Array)。这些都是常见的数据结构。事实上大部分现代计算机语言都以某种形式支持它们。这使得一种数据格式在同样基于这些结构的编程语言之间交换成为可能。

3. JSON的数据格式

JSON 具有如下这些数据格式。

1) 对象(Object)

对象(Object)是一个无序的“‘名称/值’对”集合。一个对象以“{”开始，以“}”结束。每个“名称”后跟一个“:”，“‘名称/值’对”之间使用“,”分隔。如图 13-3 所示。

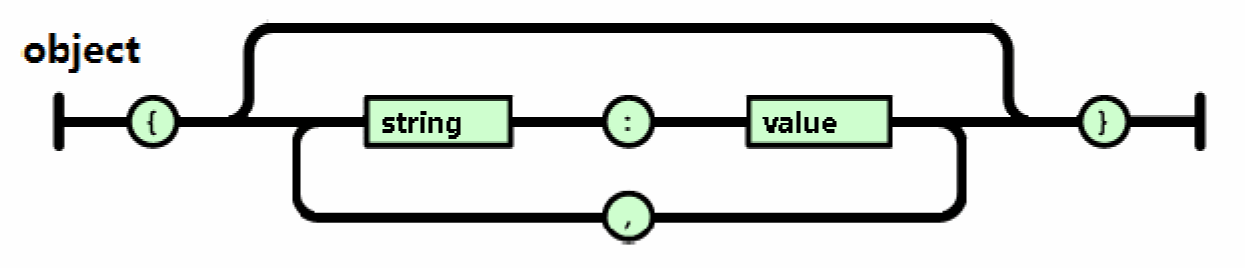


图 13-3　对象数据格式图

在 JavaScript 中使用 JSON 对象语法创建对象，代码如下所示。

```
<script type="text/javascript">
    var book = {"bookname":"Struts 2 与 Ajax 碰面","price":52};
    alert("书名: "+book.bookname+",单价: "+book.price);
</script>
```

2) 数组

数组是值的有序集合。一个数组以“[”开始，“]”结束。值之间使用“,”分隔。如图 13-4 所示。

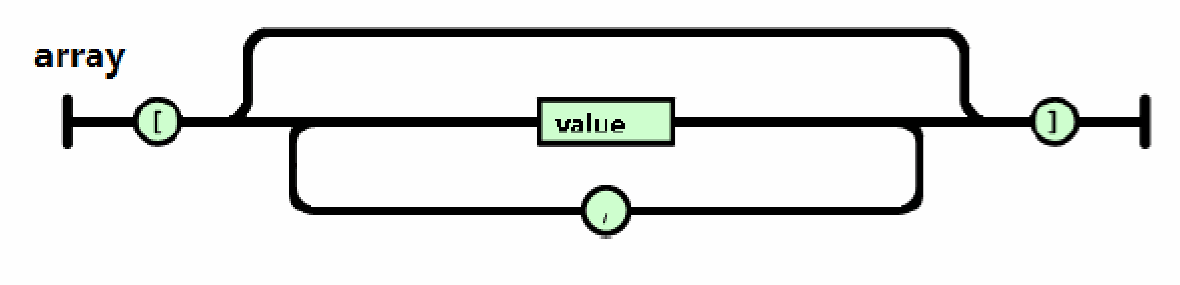


图 13-4　数组数据格式

在 JavaScript 中使用 JSON 数组语法创建数组，代码如下所示。

```
<script type="text/javascript">
    var string_array = ["你好","Hello","nihao"];
    for(var i=0;i<string_array.length;i++){
        alert(string array[i]);
    }
</script>
```

3)　值

值可以是双引号括起来的字符串、数值、true、false、null、对象或者数组。这些结构可以嵌套。如图 13-5 所示。

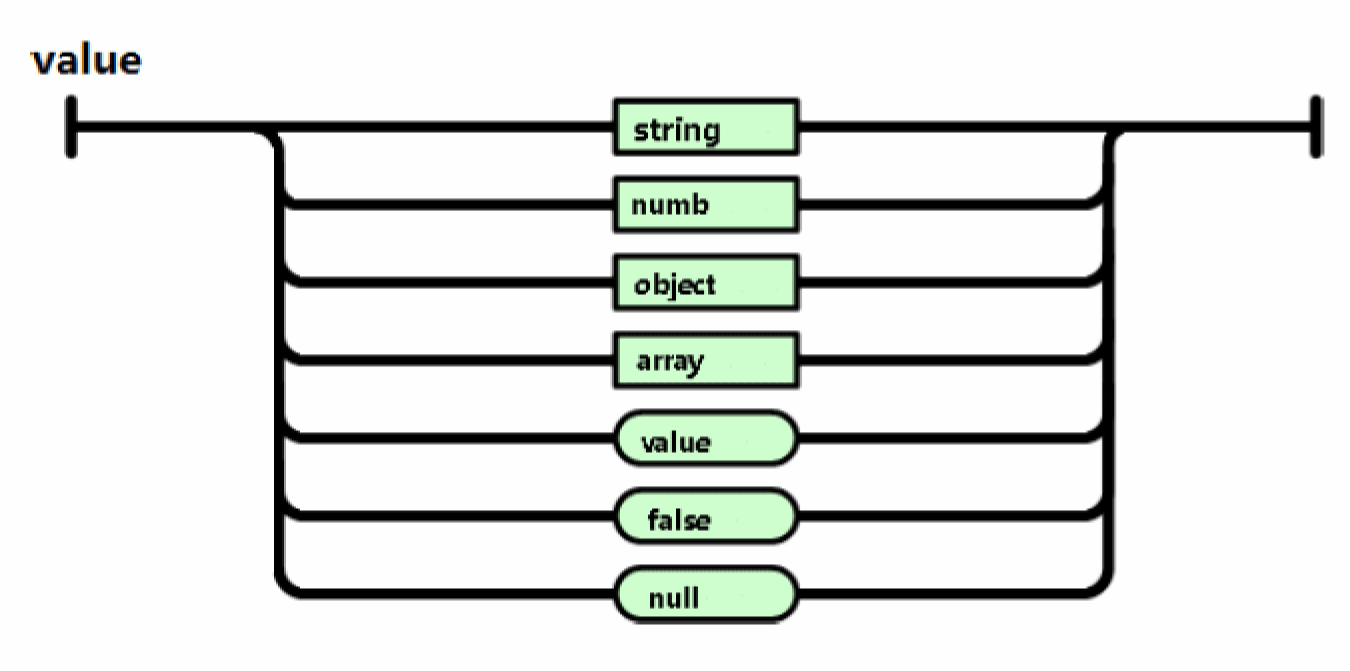


图 13-5　值的数据格式图

在数组中嵌套数值，代码如下所示。

```
<script type="text/javascript">
    var num =[10,11,12,13];
</script>
```

4)　字符串

字符串是由双引号包围的任意数量 Unicode 字符的集合，使用反斜体转义。一个字符即一个单独的字符串。如图 13-6 所示。

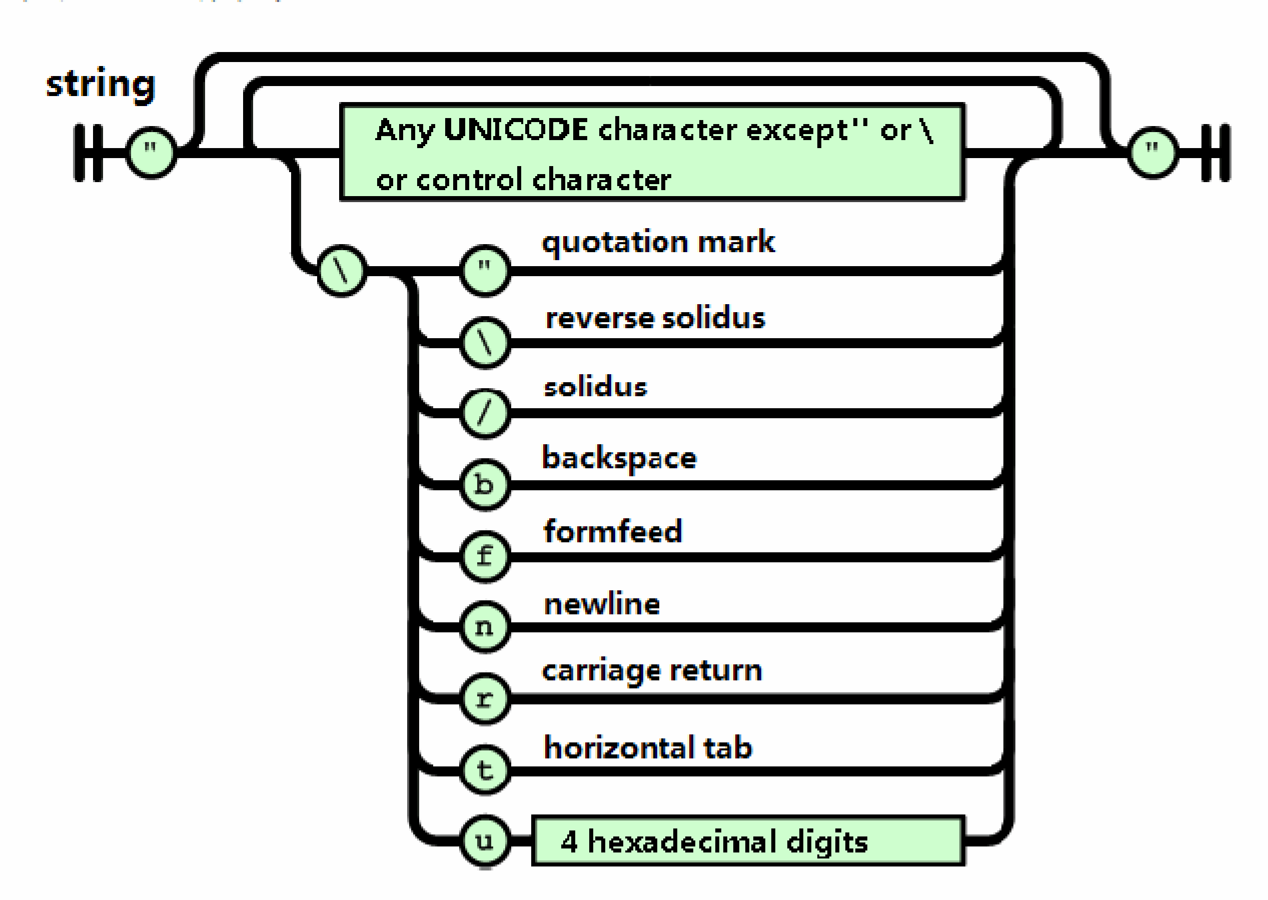


图 13-6　字符串的数据格式图

例如如下所示。

```
<script type="text/javascript">
    var str = {"welcome":"欢迎"};
    alert(str.welcome);
</script>
```

5) 数值

数值也与 C 或者 Java 中的数值非常相似，但不区分整型值和浮点型值，也不支持八进制和十六进制格式。如图 13-7 所示。

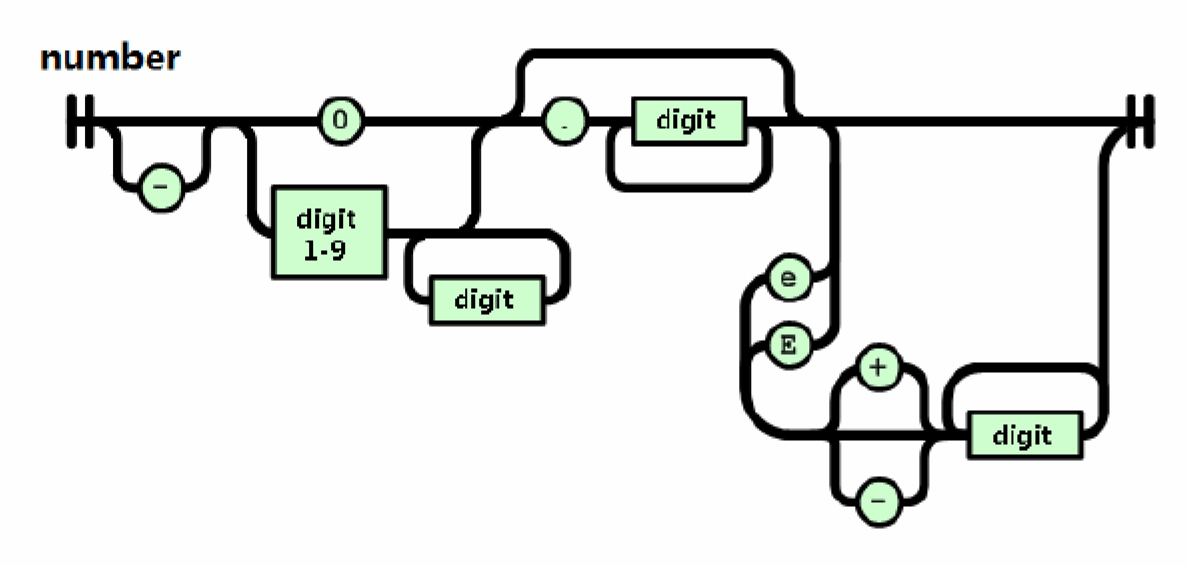


图 13-7 数值的数据格式图

学习了 JSON 的数据格式，下面我们就趁热打铁，来看一个使用 JSON 来标记地址薄的数据(包含全称、起始地址、email 地址、电话、家庭住址，等等)。代码如下所示。

```
{
    "fullname": "Sean Kelly",   //全称
    "org": "SK Consulting", //起始地址
    "emailaddrs": [          //email 地址
        {"type": "work", "value": "kelly@seankelly.biz"},
        {"type": "home", "pref": 1, "value": "kelly@seankelly.tv"}
    ],
    "telephones": [          //电话
        {"type": "work", "pref": 1, "value": "+1 214 555 1212"},
        {"type": "fax", "value": "+1 214 555 1213"},
        {"type": "mobile", "value": "+1 214 555 1214"}
    ],
    "addresses": [            //家庭住址
        {"type": "work", "format": "us",
        "value": "1234 Main StnSpringfield, TX 78080-1216"},
        {"type": "home", "format": "us",
        "value": "5678 Main StnSpringfield, TX 78080-1316"}
    ],
    "urls": [                 //url 网络连接地址
        {"type": "work", "value": "http://seankelly.biz/"},
        {"type": "home", "value": "http://seankelly.tv/"}
    ]
}
```

4. 下载使用JSON

如果要将 Java 对象和数组与 JSON 的数据结构进行交互转换操作，就需要引入 JSON 的 Java 实现 Jar 包，本书使用的是在 JSON 的官方网站“http://www.json.org/java/index.html”上下载 json.zip 包，解压该文件，可以看到 Java 源文件，将这些源文件导入到项目 src 目录下即可使用。

在 JavaScript 中也可以直接使用 JSON，因为它是 JavaScript 对象字面量语法的子集，所以只需下载一个 JavaScript 脚本文件，登录 JSON 的官方网站“https://github.com/douglascrockford/JSON-js”即可 json2.js 脚本文件，使用方式和普通的 JavaScript 脚本文件一样。



json2.jsp 文件定义了三个方法，主要用于将一个 JavaScript 对象转换成 JSON 字符串，或者将 JSON 字符串解析为 JavaScript 对象。

结合客户端的 JSON JavaScript 脚本和服务器端的 JSON Java 实现，可以将客户端 JavaScript 对象转换成 JSON 串发送到服务器端，在服务器端再将 Java 对象转换成 JSON 串发送给客户端，这样就简化数据的解析工作。

13.2.2 实例描述

前几天同事问我在 Struts 2 中可以使用 JSON 传递数据吗，我说当然可以了，他说他试了但总是接受不到数据，是空的。对此我专门抽时间深入研究了一下 JSON，并给他写了一个使用 JSON 串传递顾客信息数据的小例子，下面拿出来与读者分享学习。

13.2.3 实例应用

【例 13-2】 JSON 串传递顾客信息数据。

(1) 在项目 src/com/struts2/action 目录下，新建一个 Action 类 CustomAction，在该类中声明 customName(顾客姓名)、email(电子邮件)、address(地址)等属性，并重写。

```
package com.struts2.action;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import com.googlecode.jsonplugin.annotations.JSON;
import com.opensymphony.xwork2.ActionSupport;
import net.sf.json.JSONArray;
public class CustomAction extends ActionSupport{
    private String email;
    private String address;
    private int[] ints = { 10, 20 };
    private Map map = new HashMap();
    private String customName = "custom";
```

```
    //下面是属性 email、address、ints、map 的 get 和 set 方法，在此省略了。
    //…………………………………………………………………………………….

    @JSON(name="newName")
    public String getCustomName() {
        return customName;
    }

    public void setCustomName(String customName) {
        this.customName = customName;
    }

    public String execute(){
        map.put("email", "godenvoy@126.com");
        //格式化时间
        SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        //设置 javabean 值
        CustomAction test = new CustomAction ();
        test.setCustomName("admin");
        test.setEmail("good@gmail.com");
        test.setAddress(format.format(new Date()));
        test.setMap(map);
        //格式化 javabean 里的值
        JSONArray array = JSONArray.fromObject(test);
        return SUCCESS;
    }

    /*//测试 json 数据格式输出
    public static void main(String[] args) {
        //格式化时间
        SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        //设置 javabean 值
        CustomAction test = new CustomAction ();
        test.setCustomName("admin");
        test.setEmail("good@gmail.com");
        test.setAddress(format.format(new Date()));
        //格式化 javabean 里的值
        JSONArray array = JSONArray.fromObject(test);
        System.out.println(array.toString());
    } */
}
```

(2) 打开项目 src 目录下的 struts.xml 文件，在该文件中添加 CustomAction 配置信息，代码如下所示。

```
<package name="json" extends="json-default">
    <action name="example" class="com.struts2.action.CustomAction">
        <result type="json"/>
```

```
    </action>
</package>
```

(3) 新建一个 JSP 页面 json.jsp，在该页面中创建一个顾客信息表单，当用户单击“提交”按钮提交表单时，将触发执行 gotClick()函数。

gotClick()函数把用户输入的 customName(顾客姓名)、email(电子邮件)和 address(家庭住址)，作为参数传递给 custom.action，并指定回调函数 onComplete:processResponse，该回调函数再将 Action 处理后，返回的 JSON 格式的顾客信息数据，显示在弹出的对话框中。代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>struts2 json 插件</title>
        <script src="prototype.js" type="text/javascript"></script>
        <script type="text/javascript">
            function gotClick(){
                var customName = $("customName").value;
                var email = $("email").value;
                var address = $("address").value;
                //请求的地址
                var url = 'custom.action';
                var req = {
                    "customName":customName,
                    "email":email,
                    "address":address
                };
                //var params = Form.serialize('form1');
                //创建 Ajax.Request 对象，对应于发送请求
                var myAjax = new Ajax.Request(
                url,
                {
                    //请求方式：POST
                    method:'post',
                    //请求参数
                    //parameters:params,
                    parameters:req,
                    //指定回调函数
                    onComplete: processResponse,
                    //是否异步发送请求
                    asynchronous:true
                });
            }
            function processResponse(request){
                alert(request.responseText);
                var result = eval("("+request.responseText+")");
                alert(result+" : "+result.ints+"  "+result.email);
                $("show").innerHTML = request.responseText;
            }
```

```
        </script>
    </head>
    <body>
        <form id="form1" name="form1" method="post">
            顾客姓名：<INPUT TYPE="text" name="customName" id="customName"
/><br/>
            电子邮件：<INPUT TYPE="text" name="email" id="email" /><br/>
            家庭住址：<INPUT TYPE="text" name="address" id="address" /><br/>
            <INPUT TYPE="button" value="提交" onClick="gotClick();" />
        </form>
        <div id="show"></div>
    </body>
</html>
```

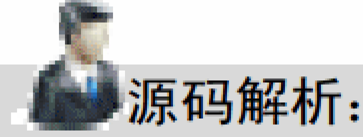
13.2.4 运行结果

打开 IE 浏览器，在地址栏中输入“http://localhost:8080/Struts2_13/json.jsp”，输入顾客信息，如：顾客姓名、电子邮件和家庭住址。单击“提交”按钮，将会在弹出对话框中，看到用户输入的顾客信息以 JSON 格式显示出来，如图 13-8 所示。



图 13-8 JSON格式传递顾客信息

13.2.5 实例分析

源码解析：

上述实例中，首先创建了一个 Action 类 CustomAction，包含 customName(顾客姓名)、email(电子邮件)和 address(家庭住址)等属性，重写了 execute()方法，在该方法中使用 JSONArray.fromObject()方法，把 CustomAction 的实例对象格式化成 JSON 格式，该 Action 以 JSON 格式返回顾客信息数据成功后，跳转到 json.jsp 页面，在该页面中弹出一个对话框来显示 JSON 格式的顾客信息数据。

13.3　Dojo异步获取用户信息

Dojo 是一个开源的 JavaScript 工具包，本身由许多模块组合而成，可以实现完整的轻量级窗口组件及很多功能。Dojo 的包加载机制(Package System)可以实现动态加载所需模块，而且用户可以编写自己的 Dojo 扩展模块，有很大的灵活性。

视频教学：光盘/videos/13/dojo.avi　　长度：8 分钟

13.3.1　基础知识——结合Dojo简化Ajax应用的开发

通过前面的学习，读者对 Struts 2 提供的 Ajax 支持应该有了一定的了解，本节将介绍一个新的 Ajax 框架——Dojo，Struts 2 对它进行了完美的封装，使用起来更加简单方便。

1. Dojo介绍

目前浏览器的种类有很多，例如 IE、FireFox、Opera 和 Safari 等。实现某个功能，如果只针对一个浏览器，代码的编写就比较简单；而如果针对不同的浏览器，就需要写不同的代码。使用 Ajax 技术时就需要考虑这样的问题，需要将不同的浏览器都考虑进去，这样就给代码的编写带来麻烦。

而 Dojo 框架能够解决这个问题。Dojo 不仅存在于抽象层，而且是独立存在的。它不只是提供一些库、方法和功能，而且能保证代码只包含所需要的部分，让代码更简洁，更有效率，并且可以更好的重复使用。



Dojo 框架是一个基于客户端的框架，利用 Dojo，可以很容易为网页或其他任何支持 JavaScript 的环境增加动态能力。

2. Dojo的特征

Dojo 大体上有 5 个特征，具体如下所示。

- 利用 Dojo 提供的组件，可以提升 Web 应用程序可用性、交互能力以及功能上的提高。
- 可以更容易地建立互动的用户界面。同时 Dojo 也提供小巧的动态处理工具。
- 利用 Dojo 的低级 API 和可兼容的代码，能够写出轻便的、单一风格(复杂)的 JavaScript 代码。Dojo 的事件系统、I/O 的 API 以及通用语言形式是基于一个强大编程环境。
- 通过 Dojo 提供的工具，可以为代码写命令行式的单元测试代码。
- Dojo 的扩展包能够使代码更容易维护，耦合性更低。

3. 下载安装Dojo工具包

使用 Dojo 框架，首先需要下载它的工具包，下面介绍 Dojo 的下载与安装。

1)　下载

打开 Dojo 的官方网站“http://download.dojotoolkit.org/”，下载页面如图 13-9 所示。

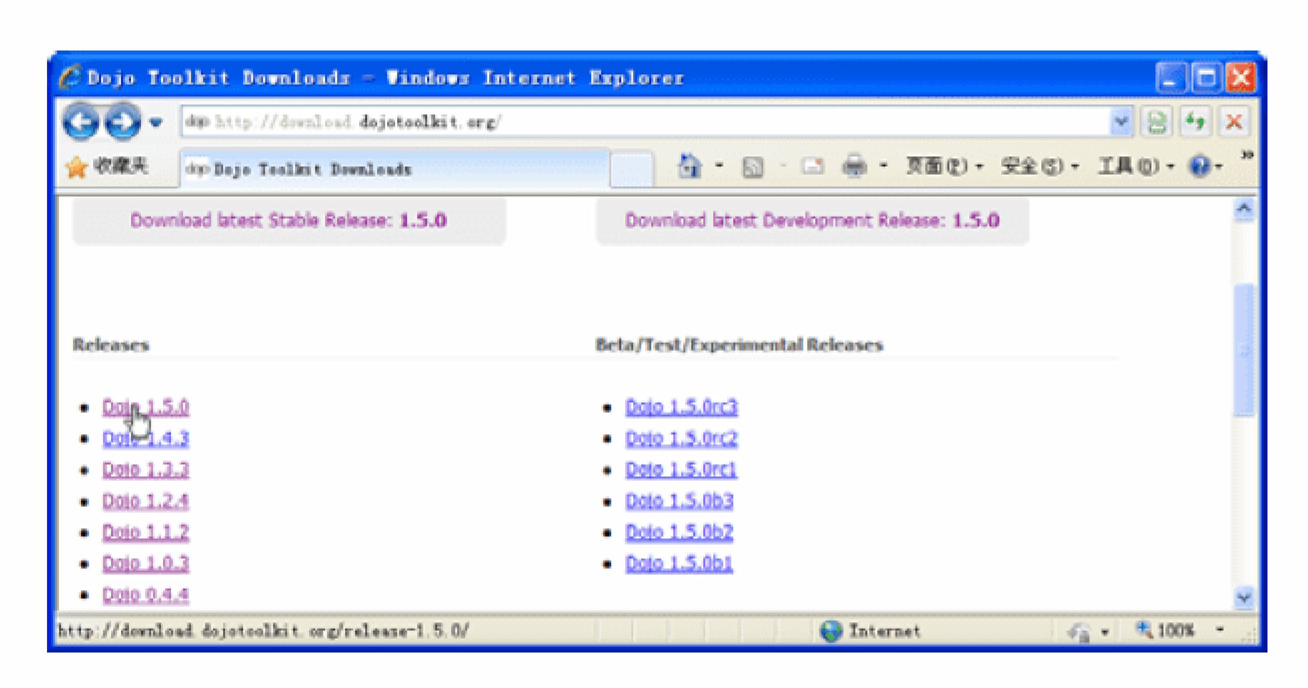


图 13-9　Dojo官网下载页面

下载最新版本 1.5.0，单击 1.5.0 下载链接，进入如图 13-10 所示的页面。单击 dojo-release-1.5.0.zip 下载链接，即可下载。

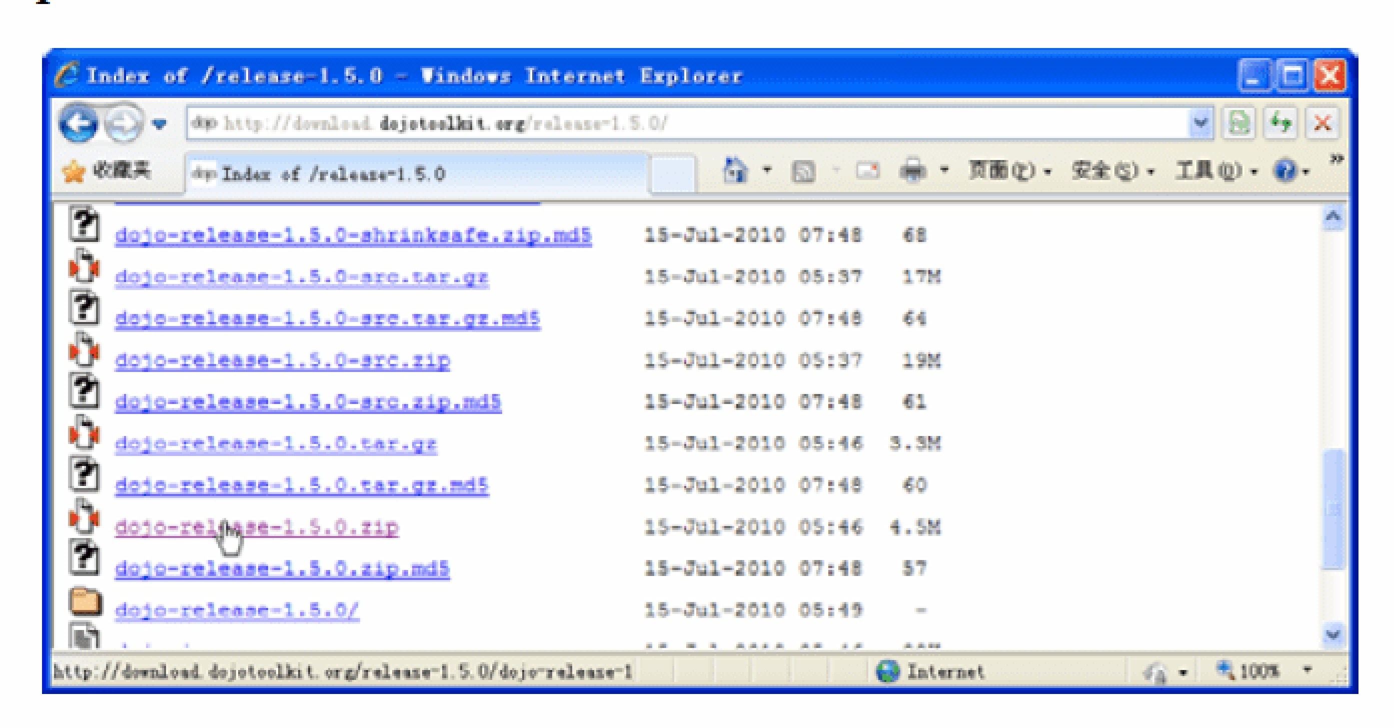


图 13-10　dojo-release-1.5.0.zip下载页面

2)　安装

下载完成之后，解压 dojo-release-1.5.0.zip 文件，在解压后的 dojo-release-1.5.0/dojo 目录下可以找到一个 dojo.js 文件，使用 Dojo 只需要将这个文件复制到项目的相应目录中的即可使用，如：在项目 WebRoot 目录下新建一个 dojo 目录，然后将 dojo.js 文件复制到 dojo 目录下。



在 Web 应用中安装好 Dojo 框架后，如果在页面使用该框架，需要添加语句：<script type="text/javascript" src="dojo/dojo.js"></script>。

4．Dojo常用函数

1)　dojo.connect

dojo.connect 用于将指定的事件处理函数绑定到事件上，也可以绑定到某个函数上，在被绑定的函数执行后，指定的函数将会被触发执行。dojo.connect 是 Dojo 的最主要的事件处理及委托方式，它可接收 5 个参数。

```
dojo.connect(obj, event, context, method, dontFix);
```

- obj：事件源对象，或被绑定函数的作用域，默认值(或 obj 被设为 null)为 dojo.global。
- event：事件名或被绑定函数名，结合第一个参数，被绑定事件(或函数)会被指定为 obj[event]。

- context：事件处理函数(绑定函数)的作用域。默认值(或 context 被设为 null)为 dojo.global。
- method：事件处理函数(绑定函数)的名称或是函数引用，如果为函数名称，结合 context 参数，该函数会被指定为 context[method]。这个函数会在事件或绑定函数执行后被触发。该函数接收的参数与事件或被绑定函数的参数相同。
- dontFix：这是一个可选参数，如果第一个参数 obj 为一个 DOM 节点对象，且 dontFix 设为 true，则事件处理不会委托给 DOM 事件管理器来进行分发。

dojo.connect 的具体用法如下所示。

```
// 当 obj.onchange()执行后，调用 ui.update():
dojo.connect(obj, "onchange", ui, "update");
dojo.connect(obj, "onchange", ui, ui.update);
```



dojo.connect 将会返回一个 handle 对象，强烈建议你在代码中保存这些 handle 对象，并在代码进行销毁时调用 dojo.disconnect 来销毁这些连接，否则将会导致内存泄露。

2)　dojo.xhr

dojo.xhr 并不是一个函数，XHR 是 XMLHTTP request object (XMLHTTP 请求对象)的一个简写。但 Dojo 封装了一系列的 XHR 函数用于 Ajax 交互，包括：dojo.xhrGet、dojo.xhrPost、dojo.xhrDelete、dojo.xhrPut、dojo.rawXhrPost、dojo.rawXhrPut 等。下面看一下 xhrGet 的具体用法，如下所示。

```
var queryObj = {sort: "id"};
var xhrHandler = dojo.xhrGet(
    {
        url: "test.jsp",
        content: queryObj
    }
);
xhrHandler.addCallback(function(data){
    datahandler(data);
});
xhrHandler.addErrback(function(error){
    errorHandler(error);
});
```

dojo.xhr*接收一个 JSON 对象作为参数，该参数可包括众多属性，以下列出一些重要的属性。

- url：XHR 的数据请求接收 URL，由于 XHR 的安全限制，该 URL 必须与脚本处于相同的域及端口下。
- timeout：超时设定，单位为 ms，当等待时间超过给定值，则会抛出一个错误给指定的错误处理回调函数。
- sync：Boolean 值，指定该 XHR 请求是同步或是异步，默认值为 false(异步)。
- content：一个 JSON 对象，其包含的键值对，将作为请求参数添加到 URL 后面。

下面来看一下它的具体用法，如下所示。

```
// 发送一个 post 请求，并忽略 response
dojo.xhrPost({
    form: "someFormId", // 在发送 Post 请求时，请求数据由 form 表单” someFormId”
    提供，URL
                    //可以通过 form 的 action 属性获取
    timeout: 3000,
    content: { part:"one", another:"part" }    //
creates ?part=one&another=part
});
// 获取 JSON 数据
dojo.xhrGet({
    url:"data.json",
    handleAs:"json",
    load: function(data){
        for(var i in data){
            console.log("key", i, "value", data[i]);
        }
    }
})
```

3) dojo.subscribe

dojo.subscribe 用于注册函数监听某个被发布的频道。当调用 dojo.publish 来向被监听的频道发送数据时，被注册的监听函数则被触发，接收发送的数据作为参数。它的语法格式如下。

```
dojo.subscribe(channel, function);
```

- channel：发布数据的频道。
- function：注册的监听函数。

dojo.subscribe 的具体用法如下所示。

```
// 注册监听函数，监听"/foo/bar/baz"频道
dojo.subscribe("/foo/bar/baz", function(data){
    console.log("i got", data);
});
// 当想触发被注册的监听函数时，调用 dojo.publish 向"/foo/bar/baz"频道发送数据
dojo.publish("/foo/bar/baz", [{ some:"object data" }]);
// 这些频道的名字可以为任意定义的字符串
dojo.subscribe("foo-bar", function(data){ /* handle */ });
```

4) dojo.connectPublisher

dojo.connectPublisher 用于绑定某些事件，以便自动向指定的频道发布数据。它的语法格式如下。

```
dojo.connectPublisher(channel, object, event);
```

- channel：发布数据的频道。
- object：事件源对象，或被绑定函数的作用域。
- event：绑定事件的名称。

dojo.connectPublisher 的具体用法如下所示。

```
dojo.connectPublisher("/some/topic/name",myObject,"myEvent");
```

13.3.2　实例描述

通过前面的学习，相信读者已经感觉到 Ajax 技术确实给用户带来了很好的体验，下面将看看另一个秘密武器——Dojo 框架，它是一个基于客户端的框架，对各种浏览器都提供了很好的支持。利用 Dojo，可以很容易地为网页或其他任何支持 JavaScript 的环境增加动态能力。比如，异步获取一些数据信息。本节实例将展示异步获取用户信息这一功能。

13.3.3　实例应用

【例 13-3】 Dojo 异步获取用户信息。

(1)　在项目 src/com/pojo 目录下，新建一个 User 实体类，声明 id、name、email 和 address 属性，并分别给出它们的 get 和 set 方法，代码如下所示。

```
package com.pojo;

public class User {
    private int id;          //编号
    private String name;     //姓名
    private String email;    //电子邮件
    private String address; //家庭住址

    //下面是 id、name、email 和 address 属性的 get 和 set 方法，在此省略了。
    //........................................................
}
```

(2)　新建一个 JSP 页面 dojo.jsp，在该页面中使用 dojo.xhrGet()函数从 getUser.jsp 页面中获取用户信息，代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>My JSP 'dojo.jsp' starting page</title>
    </head>
    <body>
        <h1>Dojo 框架异步通信</h1><br/>
        <div id="showDate"></div><br/>
        <input type="button" value="获取用户信息" onclick="getUser()"/>
        <script type="text/javascript" src="dojo/dojo.js"></script> <!-- 引
用 Dojo 框架-->
        <script type="text/javascript">
```

```
            function getUser(){              //按钮单击事件调用函数
                dojo.xhrGet(
                    {
                        url:"getUser.jsp",       //指定请求链接
                        load:returnDate,         //指定回调函数
                        error:dealError      //指定错误处理函数
                    }
                );
            }
            function returnDate(data,ioArgs){       //回调函数
                document.getElementById("showDate").innerHTML = data;
            }
            function dealError(data,ioArgs){        //错误处理函数
                document.getElementById("showDate").innerHTML = "服务器访问失
败！";
            }
        </script>
    </body>
</html>
```

(3) 新建一个 JSP 页面 getUser.jsp，在该页面中使用<s:bean>标签引入实体类 User 对象，并对其属性赋值，再使用<s:property>标签输出用户信息数据，代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>显示用户信息</title>
    </head>
    <body>
        <s:bean name="com.pojo.User" id="user">
            <s:param name="name" value="'刘晓波'"/>
            <s:param name="email" value="'liuxiaobo@sina.cn'"/>
            <s:param name="address" value="'浙江绍兴'"></s:param>
        </s:bean>
        <p>
            姓名：<s:property value="#user.name"/><br/>
            电子邮件：<s:property value="#user.email"/><br/>
            家庭住址：<s:property value="#user.address"/>
        </p>
    </body>
</html>
```

13.3.4 运行结果

运行实例，打开 IE 浏览器，在地址栏中输入“http://localhost:8080/Struts2_13/dojo.jsp”，进入显示用户信息页面，单击“获取用户信息”按钮，可以获取用户信息并显示在页面上，执

行效果如图 13-11 所示。

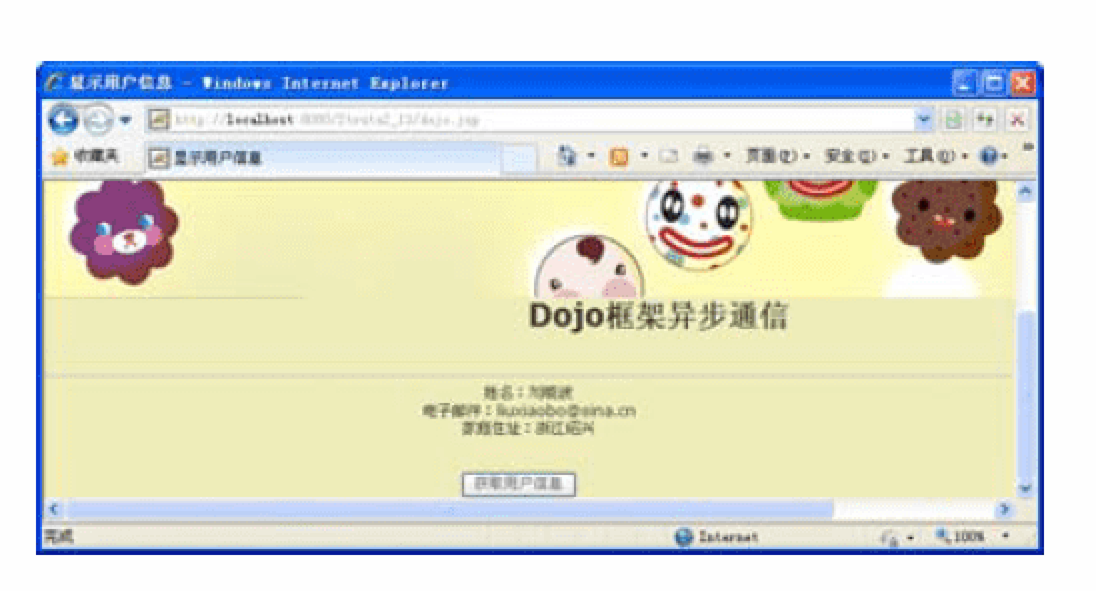
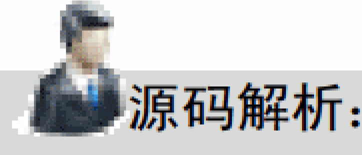


图 13-11　Dojo异步获取用户信息

13.3.5　实例分析


源码解析：

本实例首先创建一个 User 用户实体类，声明了 id、name、email 和 address 属性，然后新建了一个 JSP 页面 dojo.jsp，在该页面中添加了一个“获取用户信息”按钮，为该按钮添加了一个 onclick 鼠标单击事件，触发执行 getUser()函数，该函数使用 dojo.xhrGet()函数从 getUser.jsp 页面中获取用户信息。最后新建 getUser.jsp 页面，在该页面中使用<s:bean>标签实例化 User 实体类，使用<s:param>标签为 User 对象属性赋值，并使用<s:property>标签在页面中输出显示 User 对象属性值。

13.4　Ajax的异步请求来获取服务端数据

鉴于 Ajax 的强大交互功能，对于一个成功的 Web 框架来说，简易的 Ajax 集成是不可或缺的。Struts 2 作为一款优秀的基于 MVC 的 Java Web 框架，提供了比较完善的 Ajax 的支持。前面已经介绍了 Struts 2 对两个比较成熟的 Ajax 框架(Dojo 和 DWR)的支持，另外 Struts 2 还提供了一系列常用的 Ajax 标签，使 Ajax 开发更简便。

视频教学： 光盘/videos/13/ajax.avi　　长度：8 分钟
光盘/videos/13/a.avi　　长度：9 分钟
光盘/videos/13/div1.avi　　长度：13 分钟
光盘/videos/13/div2.avi　　长度：9 分钟
光盘/videos/13/tabbedPanel.avi　　长度：6 分钟
光盘/videos/13/autocompleter.avi　　长度：11 分钟

13.4.1　基础知识——Struts 2 的Ajax标签

Struts 2 提供的一些常用 Ajax 标签可以满足普通的 Ajax 需要，比如：div 标签、submit 标

签、a 标签，等等。本节将详细讲解这些标签的使用。

1．div标签

div 标签指<s:div>标签，它可以在页面中生成一个 div 元素，但它生成的 div 元素与我们以往使用的 div 元素不同的是，这里的 div 标签的内容是通过 Ajax 的异步请求来获取的，以实现局部内容的刷新。

1) div 标签属性

首先介绍一下 div 标签的主要属性，如表 13-1 所示。

表 13-1 div标签的属性

名　称	说　明
afterLoading	指定获取内容后需要执行的 JavaScript 代码
afterNotifyTopics	指定在请求之后(如果请求成功)发表的话题清单，话题之间使用“,”分开
autoStart	指定页面加载后是否自动启动定时器
beforeNotifyTopics	指定在请求之前发表的话题清单，话题之间使用“,”分开
closable	指定当使用 div 标签作为选项卡的一个 Tab 页面时，是否显示关闭按钮
delay	指定更新内容的时间延迟，单位为 ms。如果不指定此属性，则页面在加载后就获取数据。如果指定此属性，同时也指定 updateFreq 属性，则页面加载后需要先度过延迟时间，然后获取数据(延迟效果不会超出更新时间)；如果没有指定 updateFreq 属性，则此属性无实际意义
errorNotifyTopics	指定在请求之后(如果请求失败)发表的话题清单，话题之间使用“,”分开
errorText	指定获取数据发生错误时的提示信息
executeScripts	指定是否在本页面执行服务器响应的 JavaScript 脚本代码，其默认值为 false
formFilter	指定过滤表单字段的函数
formId	指定表单的 Id，表单的字段将被序列化并作为参数传递
handler	指定本页面的脚本函数作为处理函数。如果指定了此属性，则不会向服务器发送 Ajax 请求
highlightColor	指定突出显示颜色，对 targets 属性所指定的元素进行突出显示
highlightDuration	指定 targets 所指定元素进行突出的持续时间，单位为 ms。如果 highlightColor 属性无值，此属性无效
href	指定动态获取服务器端数据的 URL
indicator	指定动态加载服务器端数据过程中的显示内容，这里一般指定图标
javascriptTooltip	指定是否使用 JavaScript 生成浮动提示框
listenTopics	指定触发远程调用的话题
loadingText	指定当处理正在处理时显示的文本，如果异步请求发生错误则错误信息将显示在 div 内容中，如果不想显示错误信息，可以将 showErrorTransportText 属性设置为 false，如果想定制这个错误消息，可以使用 errorText 属性
notifyTopics	指定在请求之前、请求之后以及发生错误时发表的话题清单，话题之间使用“,”分开

续表

名　称	说　明
openTemplate	打开 HTML 文件的显示模板
parseContent	指定是否分析返回的动态 Web 内容以寻找组件
preload	指定是否在加载页面的同时加载动态 Web 内容
refreshOnShow	指定是否需要在 div 元素变得可见时加载动态 Web 内容。此属性在 div 元素包含于 tabbedpanel 元素中时有效
separateScripts	指定是否需要为每个标签单独创建一个范围来运行脚本代码
showErrorTransportText	指定是否显示错误信息
showLoadingText	指定是否在装载内容时显示提示信息
startTimerListenTopics	指定一个监听的事件主题，当 Struts 2 组件向该主题发布事件时，div 标签的计时器自动启动
stopTimerListenTopics	指定一个监听的事件主题，当 Struts 2 组件向该主题发布事件时，div 标签的计时器自动停止
transport	用来传递相关请求参数的传输对象
updateFreq	指定内容的更新时间间隔，单位为 ms。如果不指定此属性，则内容只有在页面加载时才会更新
refreshListenTopic	指定主题名，当该主题事件发布时，div 内容将重载

2)　div 标签应用

通过上面的学习，读者对 div 标签的属性有了基本的了解，下面运用一个小例子来帮助读者对 div 标签属性应用的理解。

(1) 首先看一个 DateAction，声明一个 date 属性，并重写 ActionSupport 的 execute()方法，用于获取系统当前时间，代码如下所示。

```
public class DateAction extends ActionSupport{
    private Date date;

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public String execute() throws Exception
    {
      date = new Date();
      return SUCCESS;
    }
}
```

(2) 编写完 DateAction 之后，还需要在 struts.xml 文件中，添加入它的配置信息，代码如下所示。

```
<package name="div" extends="struts-default">
    <action name="date" class="com.gdupt.action.DateAction">
        <result>/showDate.jsp</result>
    </action>
</package>
```

(3) 然后，在项目中新建一个 JSP 页面 showDate.jsp，用于显示当前系统时间，当 DateAction 执行成功之后将跳转到该页面，代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head></head>
    <body>
        服务器的当前时间为：<font color="red">
        <s:date name="date" format="yyyy-MM-dd HH:mm:ss"/></font>
    </body>
</html>
```

(4) 最后，在项目中新建一个 JSP 页面 div.jsp，通过为 div 标签指定 href="%{date}"。其中 date 是某个 url 标签的 id 属性值，该 url 标签的 value 属性值指定了获取数据的 DateAction。使用 div 标签动态从服务器端获取数据，代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>struts 2 ajax s:div</title>
        <s:head theme="ajax"/>
    </head>
    <body>
        <center>
            <br/><br/>
            <h5 style="color: red;">struts2 s:div 实现 AJAX 效果</h5>
            <br/>
            <s:url id="T" value="date.action" />
            在页面加载时获取数据：
            <s:div id="norefresh" theme="ajax" href="%{T}"></s:div>
            每 2 秒钟刷新一次
            <s:div id="refresh" theme="ajax" href="%{T}"
updateFreq="2000"></s:div>
            每 5 秒钟刷新一次，但有 2 秒延迟
            <s:div name="refreshD" theme="ajax" href="%{T}" updateFreq="5000"
delay="2000"/>
        </center>
    </body>
</html>
```

运行程序，打开 IE 浏览器，在地址栏中输入“http://localhost:8080/Struts2_13/div.jsp”，执行效果如图 13-12 所示。



图 13-12　获取系统当前时间

从图 13-5 中可以看出第一个 div 元素与第二个 div 元素的时间相差 2 秒钟，因为第一个 div 元素没有设置 updateFreq 属性进行刷新，而第二个 div 元素是每 2 秒钟刷新一次。第 3 个 div 元素由于设置了 delay 属性，添加了时间延迟效果，还没有获取到服务器端数据。

2. submit标签

submit 标签即<s:submit>标签，它用于向服务器异步提交数据，也可以使用异步请求返回的文本来更新 HTML 元素(通常指 div)的内容。

1)　submit 标签属性

submit 标签的常用属性如表 13-2 所示。

表 13-2　submit标签的属性

名　称	说　明
errorText	指定获取数据发生错误时的提示信息
executeScripts	指定是否在本页面执行服务器响应的 JavaScript 脚本代码，其默认值为 false
formFilter	指定过滤表单字段的函数
formId	指定请求参数的表单
afterNotifyTopics	指定在请求之后(如果请求成功)发表的话题清单，话题之间使用“,”分开
ajaxAfterValidation	指定如果验证成功，是否发出一个异步请求。此属性只在 validation 属性值为 true 时有效
beforeNotifyTopics	指定在请求之前发表的话题清单，话题之间使用“,”分开
errorNotifyTopics	指定在请求之后(如果请求失败)发表的话题清单，话题之间使用“,”分开
handler	指定本页面的脚本函数作为处理函数。如果指定了此属性，则不会向服务器发送 Ajax 请求
highlightColor	指定突出显示颜色，对 targets 属性所指定的元素进行突出显示

续表

名　称	说　明
highlightDuration	指定 targets 所指定元素进行突出的持续时间，单位为 ms。如果 highlightColor 属性无值，此属性无效
href	指定动态获取服务器端数据的 URL
indicator	指定动态加载服务器端数据过程中的显示内容，这里一般指定图标
javascriptTooltip	指定是否使用 JavaScript 生成浮动提示框
listenTopics	指定触发远程调用的话题
loadingText	指定内容正在装载过程中的提示信息，主要用来提示用户正在装载内容
method	对应 HTML submit 元素的 method 属性
notifyTopics	指定在请求之前、请求之后以及发生错误时发表的话题清单，话题之间使用“,”分开
parseContent	指定是否分析返回的动态 Web 内容以寻找组件
separateScripts	指定是否需要为每个标签单独创建一个范围来运行脚本代码
showErrorTransportText	指定是否显示错误信息
showLoadingText	指定是否在装载内容时显示提示信息
src	指定当按钮类型为 Image 时，按钮的图片来源
targets	指定内容将被更新的元素清单，元素之间使用“,”分开
transport	指定用来传递相关请求的传输对象
type	指定提交按钮的类型，可选值有：input、image 和 button
validate	指定是否进行 Ajax 验证

2)　submit 标签应用

下面是一个 submit 标签应用的例子，通过 href 属性来指定要异步请求的资源地址，通过 targets 属性来指定服务器返回的响应内容应该更新哪些 HTML 元素的内容，代码如下所示。

```
<div id="divdate">这里是初始内容</div>
    <s:url id="date" value="date.action" />
    <br/>
<s:submit type="submit" theme="ajax" href="%{date}" targets="divdate"
align="left"
 value="更新指定元素的内容"></s:submit>
```

3. a标签

a 标签即<s:a>标签，用来生成一个超链接，向服务器异步提交数据，并将返回内容加载到指定页面元素中。

1)　a 标签属性

a 标签的常用属性如表 13-3 所示。

表 13-3　a标签的属性

名　称	说　明
afterNotifyTopics	指定在请求之后(如果请求成功)发表的话题清单，话题之间使用“,”分开
ajaxAfterValidation	指定如果验证成功，是否发出一个异步请求。此属性只在 validation 属性值为 true 时有效
beforeNotifyTopics	指定在请求之前发表的话题清单，话题之间使用“,”分开
errorNotifyTopics	指定在请求之后(如果请求失败)发表的话题清单，话题之间使用“,”分开
errorText	指定获取数据发生错误时的提示信息
executeScripts	指定是否在本页面执行服务器响应的 JavaScript 脚本代码，其默认值为 false
formFilter	指定过滤表单字段的函数
formId	指定请求参数的表单
handler	指定本页面的脚本函数作为处理函数。如果指定了此属性，则不会向服务器发送 Ajax 请求
highlightColor	指定突出显示颜色，对 targets 属性所指定的元素进行突出显示
highlightDuration	指定 targets 所指定元素进行突出的持续时间，单位为 ms。如果 highlightColor 属性无值，此属性无效
href	指定动态获取服务器端数据的 URL
indicator	指定动态加载服务器端数据过程中的显示内容，这里一般指定图标
javascriptTooltip	指定是否使用 JavaScript 生成浮动提示框
listenTopics	指定触发远程调用的话题
loadingText	指定内容正在装载过程中的提示信息，主要用来提示用户正在装载内容
notifyTopics	指定在请求之前、请求之后以及发生错误时发表的话题清单，话题之间使用“,”分开
openTemplate	用来打开 HTML 文件的显示模式
parseContent	指定是否分析返回的动态 Web 内容以寻找组件
separateScripts	指定是否需要为每个标签单独创建一个范围来运行脚本代码
showErrorTransportText	指定是否显示错误信息
showLoadingText	指定是否在装载内容时显示提示信息
targets	指定内容将被更新的元素清单，元素之间使用“,”分开
transport	指定用来传递相关请求的传输对象
validate	指定是否进行 Ajax 验证

2)　a 标签应用

前面已经说过 a 标签用于生成一个超链接，下面就使用 a 标签来获取系统当前时间，设置 href 属性来指定要异步请求的资源地址，targets 属性来指定服务器返回的响应内容应该更新哪些 HTML 元素的内容。代码如下所示。

```
<s:url id="date" value="date.action" />
<div id="divdate"></div>
```

```
<br/>
<s:a id="a1" theme="ajax" href="%{date}" targets="divdate">
单击此链接，从服务器端获取数据，更新 id 为 divdate 的 div 元素内容</s:a>
```

4. tabbedPanel标签

tabbedPanel 标签即<s:tabbedPanel>标签，用于生成一个 Panel(包含标签页 tab)，tab 页上的内容可以是静态的也可以是动态异步加载的。每个标签页都是一个 Ajax 主题下的 div 标签，因此作为标签页使用的 div 标签只能在 tabbedPanel 标签的内部使用，还需要使用 div 标签的 label 属性来制定标签页的标题。

1) tabbedPanel 标签

tabbedPanel 标签的常用属性如表 13-4 所示。

表 13-4 tabbedPanel标签的常用属性

名 称	说 明
ajaxAfterValidation	指定如果验证成功，是否发出一个异步请求。此属性只在 validation 属性值为 true 时有效
beforeNotifyTopics	指定在请求之前发表的话题清单，话题之间使用“,”分开
closeButton	指定 Tab 页面上关闭按钮的位置，可选值有 tab 和 pane
doLayout	指定 tabbedPanel 是否为固定高度，如果此属性值为 false，则其高度随着 Tab 页面的大小而改变
errorNotifyTopics	指定在请求之后(如果请求失败)发表的话题清单，话题之间使用“,”分开
errorText	指定获取数据发生错误时的提示信息
executeScripts	指定是否在本页面执行服务器响应的 JavaScript 脚本代码，其默认值为 false
formFilter	指定过滤表单字段的函数
formId	指定请求参数的表单
handler	指定本页面的脚本函数作为处理函数。如果指定了此属性，则不会向服务器发送 Ajax 请求
href	指定动态获取服务器端数据的 URL
indicator	指定动态加载服务器端数据过程中的显示内容，这里一般指定图标
labelposition	指定 Tab 页面中标签的位置，可选值有：top(默认值)、right、bottom 和 left
loadingText	指定内容正在装载过程中的提示信息，主要用来提示用户正在装载内容
method	对应 HTML submit 元素的 method 属性
selectedTab	指定加载该页面时，初始状态下显示哪个 Tab 页面，默认显示第一个
showErrorTransportText	指定是否显示错误信息
showLoadingText	指定是否在装载内容时显示提示信息
targets	指定内容将被更新的元素清单，元素之间使用“,”分开
transport	指定用来传递相关请求的传输对象
validate	指定是否进行 Ajax 验证



一定要将 tabbedPanel 标签的 theme 属性设置为 simple 主题，不要设置为 ajax，因为 tabbedPanel 标签在 simple 主题中，本身没有包含任何的 Ajax 功能，但是 div 标签的属性一定要设置为 ajax。

2)　tabbedPanel 标签的应用

使用 tabbedPanel 标签实现一个静态标签页，代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>我的静态 Tab 页实例</title>
        <s:head theme="ajax" debug="true"/>
        <link rel="stylesheet" type="text/css" href="<s:url
value="/struts/tabs.css"/>">
    </head>
    <body>
        <s:tabbedPanel id="mystatic" theme="simple"
cssStyle="width:300px;height:200px;">
            <s:div id="static1" label="Tab 1" theme="ajax">
                我的第一个静态标签页
            </s:div>
            <s:div id="static2" label="Tab 2" theme="ajax">
                我的第二个静态标签页
            </s:div>
        </s:tabbedPanel>
    </body>
</html>
```

5. autocompleter标签

autocompleter 标签指<s:autocompleter>标签，用来在页面中生成一个带下拉按钮的单行文本框。在页面加载时，生成像搜索引擎一样的自动向用户提示一些关键字选项。下拉列表用来向用户显示输入提示，提供可选值，它的可选值会随着用户在文本框中的输入内容的改变来给出建议性选项，当然，可选值在页面加载时生成的内容范围之内。



一定要将 tabbedPanel 标签的 theme 属性设置为 simple 主题，不要设置为 ajax，因为 tabbedPanel 标签在 simple 主题中，本身没有包含任何的 Ajax 功能，但是 div 标签的属性一定要设置为 ajax。

1)　autocompleter 标签

autocompleter 标签的常用属性如表 13-5 所示。

表 13-5　autocompleter标签的属性

名　称	说　明
afterNotifyTopics	指定在请求之后(如果请求成功)发表的话题清单，话题之间使用“,”分开
autoComplete	指定是否在单行文本框中显示输入提示
beforeNotifyTopics	指定在请求之前发表的话题清单，话题之间使用“,”分开
dataFieldName	被返回的 JSON 对象里，包含着数据数组的那个字段的名字
delay	指定搜索可选值之前的延迟时间，单位为 ms
dropdownHeigh	指定下拉列表的高度
dropdownWidth	指定下拉列表的宽度，默认与单行文本框一致
emptyOption	指定是否插入一个空选项
errorNotifyTopics	指定在请求之后(如果请求失败)发表的话题清单，话题之间使用“,”分开
forceValidOpion	指定单行文本框是否接受下拉列表中的选项
formFilter	指定过滤表单字段的函数
formId	指定请求参数的表单
headerKey	指定选项清单里的第一项的键
headerValue	指定选项清单里的第一项的值
href	指定动态获取服务器端数据的 URL
iconPath	指定下拉列表的图标文件路径
indicator	指定动态加载服务器端数据过程中的显示内容，这里一般指定图标
javascriptTooltip	指定是否使用 JavaScript 生成浮动提示框
keyName	指定将被选中的键赋给哪一个属性
list	指定生成下拉列表中选项的集合
listKey	指定列表中用来提供选项标号的对象的属性
listValue	指定列表中用来提供选项值的对象的属性
listenTopics	指定触发远程调用的话题
loadMinimumCount	当 loadOnTextChange 属性值为 true 时，用来指定输入多少字符后，才会重新加载下拉列表的选项
loadOnTextChange	指定在单行文本框中的内容发生改变时，是否需要重新加载列表选项
maxLength	对应 HTML maxlength 属性
notifyTopics	指定在请求之前、请求之后以及发生错误时发表的话题清单，话题之间使用“,”分开
preload	指定是否在加载页面的同时重新加载清单
resultsLimit	指定选项最多可以有多少个，如果此属性值为-1，表示无限制
searchType	指定下拉列表与单行文本框的字符匹配模式，可选值有：startstring(默认值，显示以文本框中字符串开头的选项)、startword(显示以文本框中单词开头的选项)和substring(显示包含文本框中字符串的选项)
showDownArrow	指定是否显示下拉箭头，其默认值为 true
transport	指定用来传递相关请求的传输对象
valueNotifyTopics	指定在有一个值被选中时发表的话题清单，话题之间使用“,”分开

2)　autocompleter 标签的应用

学习了 autocompleter 标签属性之后，来看看它的具体应用，即实现一个选择自己喜欢的水果的小实例，代码如下所示。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>autocompleter 标签示例</title>
        <s:head theme="ajax" debug="true"/>
    </head>
    <body>
        从服务器端获取数据：<s:url id="data" value="data.action" />
        选择自己喜欢的水果：
        <s:autocompleter name="auto" theme="ajax" href="%{data}"/><br/>
        <hr/>
        设置 showDownArrow="false"<br/>
        选择自己喜欢的水果：
        <s:autocompleter name="auto" theme="ajax" href="%{data}"
showDownArrow="false"/><br/>
        <hr/>
        设置 list 属性获取本地 List 数据<br/>
        选择喜爱的节日：
        <s:autocompleter name="auto" theme="simple"
        list="{'春节','端午节','中秋节','七夕节'}"/><br/>
        <hr/>
    </body>
</html>
```

动态的从服务器端获取数据，需要在 struts.xml 文件中配置一个 Action，该 Action 返回 fruit.js 文件中的 JSON 格式的数据，配置代码如下所示。

```
<package name="autocompleter" extends="struts-default">
    <action name="data">
        <result>/fruit.js</result>
    </action>
</package>
```

fruit.js 文件中的 JSON 格式数据，如下所示。

```
 [
    ["蜜桔","蜜"],
    ["橙子","橙"],
    ["杨梅","杨"],
    ["苹果","苹"],
    ["香蕉","香"],
    ["柠檬","柠"]
]
```

13.4.2 实例描述

Ajax 的异步获取信息给用户浏览页面带来了很好的体验，同时 Struts 2 提供的一些 Ajax 标签也可以进行异步获取数据操作。本节实例将为读者演示 Ajax 的具体实现，即异步获取服务器端的数据。

13.4.3 实例应用

【例 13-4】 Ajax 的异步请求来获取服务端数据。

(1) 在项目 src/com/struts2/action 包下新建一个 RandomAction，用于生成一个随机数，代码如下。

```
package com.struts2.action;
import com.opensymphony.xwork2.ActionSupport;
public class RandomAction extends ActionSupport
{
    private String data;    //数据
    public String getRdmStr(){       //生成一个随机数
        String result = String.valueOf(Math.round(Math.random() * 10000));
        return data != null && !data.equals("") ? data + result : result;
    }
    public void setData(String data){
        this.data = data;
    }
    public String getData(){
        return this.data;
    }
    public String execute(){
        return SUCCESS;
    }
}
```

(2) 打开项目 src 目录下的 struts.xml 文件，添加 RandomAction 的配置，代码如下所示。

```
<package name="ajax_tab" extends="struts-default">
    <action name="random" class="com.struts2.action.RandomAction">
        <result name="success">/random.jsp</result>
    </action>
</package>
```

(3) 新建一个 JSP 页面 random.jsp，用于显示 RandomAction 产生的随机数据，代码如下所示。

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<%
    request.setAttribute("decorator", "none");
```

```
    response.setHeader("Cache-Control","no-cache"); //HTTP 1.1
    response.setHeader("Pragma","no-cache"); //HTTP 1.0
    response.setDateHeader ("Expires", 0); //prevents caching at the proxy
server
%>
服务器返回的随机数字是：<s:property value="rdmStr"/>
```

(4) 新建一个 JSP 页面 ajax_tab.jsp，在该页面中从服务器端获取 RandomAction 生成的随机数，显示在 div 元素中，通过指定 showErrorTransportText="true"，可以在 div 元素中显示系统出错提示(如：本实例中提示 Error loading 404 AjaxNoUrl.jsp)。代码如下所示。

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
    <head>
        <title>远程 Div</title>
        <s:head theme="ajax"/>
    </head>
    <body>
        <s:url id="rd" value="random.action" />
        仅一次获取服务器内容的 Div<br>
        <s:div id="div1"
            theme="ajax"
            cssStyle="border: 1px solid black;background-color:#dddddd;
            width:300px;height:40px;padding-top:8px;padding-left:20px"
            href="%{rd}">
            初始化文本
        </s:div>
        动态更新内容的 Div，每隔 1s 刷新一次(通过指定 updateFreq="1000")<br>
        使用 indicator(通过指定 indicator="indicator")<br>
        <s:div id="div2"
            theme="ajax"
            cssStyle="border: 1px solid black;background-color:#dddddd;
            width:300px;height:40px;padding-top:8px;padding-left:20px"
            href="%{rd}"
            updateFreq="1000"
            indicator="indicator">
            初始化文本
        </s:div>
        <img id="indicator"
src="${pageContext.request.contextPath}/images/indicator.gif"
         alt="Loading..." style="display:none"/><br>
        3s 之后才开始更新(通过指定 delay="3000")<br>
        指定与服务器交互出错的提示(通过指定 errorText 属性)<br>
        指定与服务器交互过程中的提示(通过指定 loadText 属性)<br>
        <s:div id="div3"
            theme="ajax"
            cssStyle="border: 1px solid
black;background-color:#dddddd;width:300px;height:40px;padding-top:8px;padd
```

```
ing-left:20px"
            href="%{rd}" updateFreq="1000" delay="3000"
            errorText="加载服务器数据出错"
            loadingText="正在加载服务器内容">   <!--   使用变量来指定 URL-->
            初始化文本
        </s:div>
        指定显示系统出错提示(通过指定 showErrorTransportText="true")<br>
        <s:div id="div4"
            theme="ajax"
            cssStyle="border: 1px solid black;background-color:#dddddd;
            width:300px;height:40px;padding-top:8px;padding-left:20px"
            href="/AjaxNoUrl.jsp"
            updateFreq="1000"
            showErrorTransportText="true"
            loadingText="正在加载服务器内容">
            初始化文本
        </s:div>
    </body>
</html>
```

(5) 新建一个 JSP 页面 result.jsp，在该页面中采用本页面的 JavaScript 函数获取本地数据，不再调用远程服务器，代码如下所示。

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
    <head>
        <title>远程 Div</title>
        <s:head theme="ajax"/>
    </head>
    <script type="text/javascript">
        function handler(widget, node) {
            alert('本地 JavaScript 函数处理动态 Div');
            node.innerHTML = Math.random() > 0.4 ? "Spring2.0 宝典" :
            "轻量级 J2EE 企业应用实战";
        }
    </script>
    <body>
        <s:url id="rd" value="/random.action" />
        直接使用本页面的 JS 函数，不再调用远程服务器<br>
        <s:div id="div1"
            theme="ajax"
            cssStyle="border: 1px solid black;background-color:#dddddd;
            width:300px;height:40px;padding-top:8px;padding-left:20px"
            href="%{rd}" updateFreq="2000"
            handler="handler"><!-- 此时的 href 属性无效 -->
            初始化文本
        </s:div>
    </body>
</html>
```

13.4.4　运行结果

打开 IE 浏览器，在地址栏中输入“http://localhost:8080/Struts2_13/ajax_tab.jsp”，从服务器端远程获取数据，执行效果如图 13-13 所示。



图 13-13　从服务器端远程获取数据

既然能够远程的从服务器端获取数据，那么能不能采用本地数据呢？当然能了，可以采用 JavaScript 函数获取本地数据，在 IE 浏览器的地址栏中输入“http://localhost:8080/Struts2_13/result.jsp”获取本地数据，执行效果如图 13-14 所示。

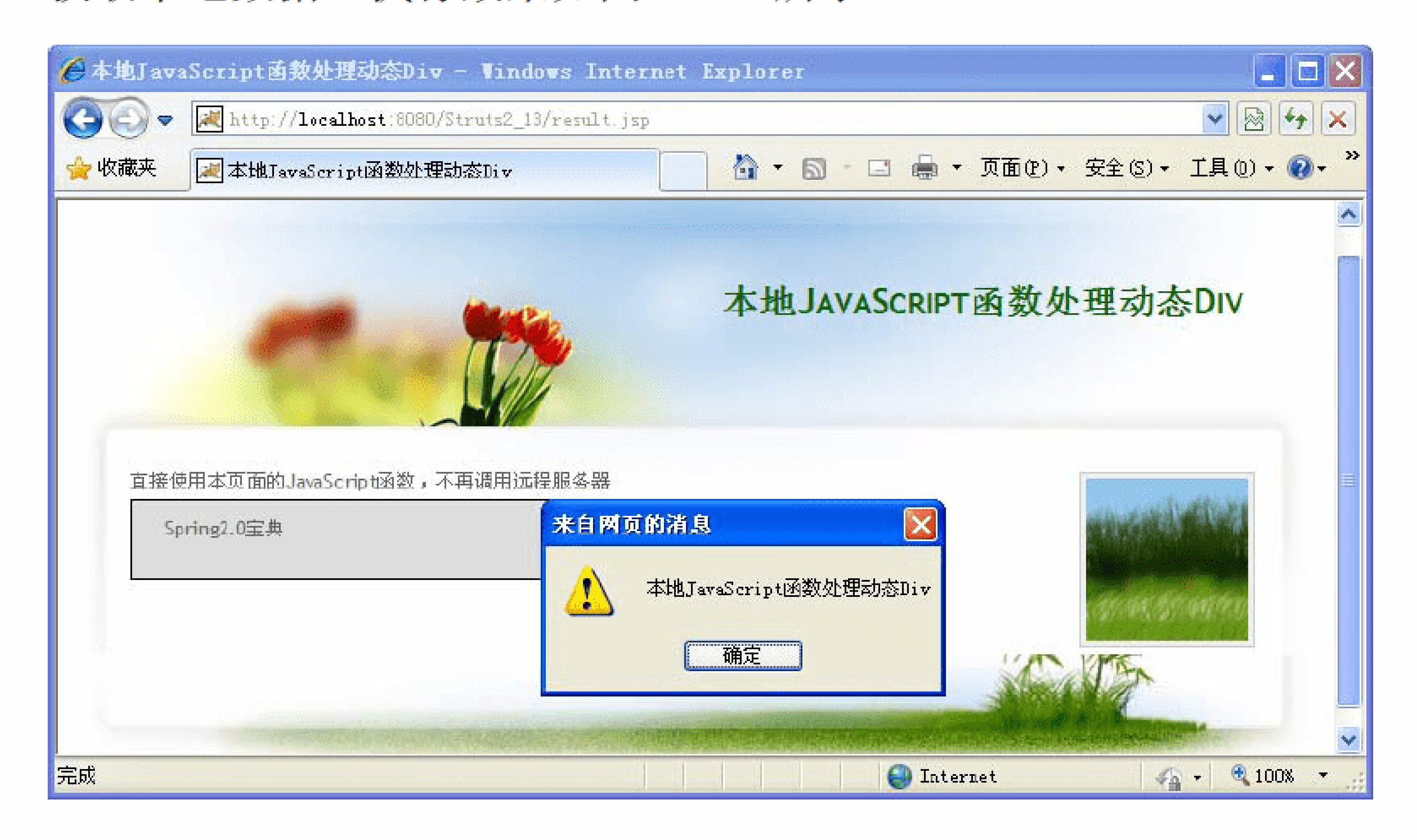


图 13-14　获取本地数据

13.4.5 实例分析



源码解析：

本实例首先创建了一个 RandomAction，生成一个随机数，当访问 ajax_tab.jsp 页面成功时，在该页面中通过<s:url>标签指定 value 属性访问 random.action，id 属性为“rd”。在<s:div>标签中通过指定 href 属性为 rd，来动态向服务器端获取数据，显示在页面上，通过指定 showErrorTransportText="true"，显示系统出错提示。

在 result.jsp 页面中，通过 div 标签的 handler 属性，指定本页面的脚本函数作为处理函数 handler。指定了此属性，则不会向服务器发送 Ajax 请求，指定的 href 属性将失效，handler 函数使用本地数据，显示在 div 元素中。

13.5 常见问题解答

13.5.1 Ajax获取Struts 2 的Action的返回信息问题



Ajax 如何获取 Struts 2 的 Action 的返回信息？

网络课堂：http://bbs.itzcn.com/thread-11016-1-1.html

我想使用 Ajax 获取 Struts 2 的 Action 返回的信息，简单的写了如下一个 check()方法，不知是不是这样写的。

```
public String check() throws Exception {
  //获取 HttpServletRequest 对象 request
  HttpServletRequest  request=ServletActionContext.getRequest();
  //获取 HttpServletResponse 对象 response
  HttpServletResponse  response=ServletActionContext.getResponse();
  response.setContentType("text/html;charset=UTF-8");
  //获取 PrintWriter 对象 out，
  PrintWriter out=response.getWriter();
  System.out.println(user.getUserName());
  User u=userService.getUserByName(user.getUserName());
  if(u==null)
  {
    //输出 jsp 页面
    out.println("<html><body>恭喜您用户名可用！</body></html>");
    return null;
  }
  else
  {
    out.println("<html><body>对不起，用户名已存在！</body></html>");
```

```
    return null;
  }
}
```

但是，输入用户名为中文时无法接收参数。求助高手。

【解决办法】：你的方法里不用写成 String，写成 void 不返回任何东西，在 out.println ("<html><body>恭喜您用户名可用！</body></html>");，前台的 Ajax 部分直接接受返回过来的 msg 信息即可。

13.5.2　Struts 2 中使用Ajax标签出错问题



Struts 2 中使用 Ajax 标签出错?
网络课堂：http://bbs.itzcn.com/thread-11019-1-1.html

今天刚学 Struts 2 使用 Ajax 标签就遇到问题了，正如标题写的，我在 JSP 页面的<head></head>之间插入<s:head theme="ajax"/>这句话，打开页面弹出一个对话框提示站点无法访问，如果把<s:head theme="ajax"/>这句取消了，才能打开页面，但这页面也失去 Ajax 功能了。环境的配置都没问题，请教高手解释一下原因出错原因。我把页面部分关键代码粘贴出来，代码如下所示。

```
<html>
    <head>
        <base href="<%=basePath%>">
        <title>My JSP 'index.jsp' starting page</title>
        <meta http-equiv="pragma" content="no-cache">
        <meta http-equiv="cache-control" content="no-cache">
        <meta http-equiv="expires" content="0">
        <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
        <meta http-equiv="description" content="This is my page">
        <s:head theme="ajax"/> <!--加了这句就打不开页面-->
    </head>
    <body>
        <s:form action="register" theme="ajax" validate="true">
            <s:textfield label="用户名" name="username"></s:textfield>
            <s:password label="密码" name="password"></s:password>
            <s:password label="重新输入密码" name="repassword"></s:password>
            <s:textfield label="年龄" name="age"></s:textfield>
            <s:textfield label="生日" name="birthday"></s:textfield>
            <s:submit value="发布"></s:submit>
        </s:form>
        <s:property value="result"/>
    </body>
</html>
```

【解决办法】：如果你的 Struts 版本是 2.1.6 以上的，那就要加 struts2-dojo-plugin-2.1.6.jar，Struts 2 升级以后把 Ajax 单独提出来放这个里面了。

我用的是 struts 2.1.8，加了 struts2-dojo-plugin-2.1.8.jar 后，页面改成如下所示。

```
<%@ taglib prefix="s" uri="/struts-tags"%>
<%@ taglib prefix="sd" uri="/struts-dojo-tags"%>
<s:head />
<sd:parseContent="true"/>
```

13.5.3 Struts 2 怎样获取Ajax post请求传递的数据？



Struts 2 怎样获取 Ajax post 请求传递的数据？
网络课堂：http://bbs.itzcn.com/thread-11021-1-1.html

我需要从页面传递一个对象数组，不知道 Struts 2 怎样获取这个数组？而且这个数组对象需要转换成一个 List 对象，提示 Ognl 无法转换的错误。请高手赐教。

【解决办法】：从页面传对象到 Action，Action 用 List 接收，这样肯定是可以的，只要你的参数名字相同就行。如果真的不行，那就是用 Struts 2 的类型转换功能，手动把 JS 数组串转换为 Java 的 List，需要用 Strutstypeconverter 接口，从 Action 传对象到页面，把 JSON 串以流的形式写入页面，这样 Ajax 就会接到。如果你不熟悉，那么可以使用 JSON 插件，目的是简化你的页面流操作，到页面的 JSON 只是字符串，通过 JS 的 exal()方法动态编译就能得到 JSON 对象。

13.6 习　　题

一、填空题

(1) 使用 DWR 框架需要配置的核心 Servlet 是__________。

(2) JSON 有两种构建结构：“名称/值”对的集合和__________。

(3) JSON 的数据格式有：对象、数组、值、__________和数值。

(4) Dojo 的常用函数 dojo.connect 主要用于______________________________。

二、选择题

(1) Ajax 术语是由__________公司或组织最先提出的。(单选)

A. Google　　B. IBM　　C. Adaptive Path　　D. Dojo Foundation

(2) 以下__________Web 应用不属于 Ajax 应用。(单选)

A. Hotmail　　B. Gmaps　　C. Flickr　　D. Windows Live

(3) 以下__________技术不是 Ajax 技术体系的组成部分。(单选)

A. XMLHttpRequest　　B. DHTML　　C. CSS　　D. DOM

(4) 下列__________方法或属性是 Web 标准中规定的。(单选)

A. all()　　B. innerHTML

C. getElementsByTagName()　　D. innerText

(5) 下列__________工具不能用来调试浏览器中的 JavaScript。(单选)

A. MS Visual InterDev　　B. Eclipse

C. MS Script Debugger　　D. Mozilla Venkman

(6) 关于 JavaScript 中的函数和对象，下列说法不正确的是__________。(单选)

A. 每一个函数都有一个 prototype 对象。

B. 函数就是一个特殊类型的对象。

C. 函数附属于它所附加到的对象上，只能通过该对象访问。

D. 同一个函数可以被附属到多个对象上。

(7) 创建一个对象 obj，该对象包含一个名为“name”的属性，其值为“value”。以下哪一段 JavaScript 代码无法得到上述的结果？__________(单选)

A.
```
var obj = new Object();
obj["name"] = "value";
```
B.
```
var obj = new Object();
obj.prototype.name = "value";
```
C.
```
var obj = {name : "value"};
```
D.
```
var obj = new function() {
    this.name = "value";
}
```

三、上机练习

上机练习：省市级联。

要求：首先需要创建 Province 和 City 两个实体类，然后创建 CityDao 操作数据库类，定义查询省方法 findAllProvince()，通过省编号查询省方法 findProvinceById(int proId)，通过省编号查询该省的所有城市方法 findCitysByProId(int proId)，通过城市编号查询城市方法 findCityById(int cityId)。接下来再创建一个 Action 类 ShowCity.java，在该类中创建四个执行方法，分别是 findAllProvince()获取所有省数据、findProvinceById()通过省编号获取省数据、findCitysByProId()通过省编号获取该省的所有城市数据和 findCityById()通过城市编号获取城市数据。最后创建一个 JSP 页面 showCity.jsp，在该页面中定义两个 select 标签分别用于选择省和城市。

执行效果如图 13-15 所示。

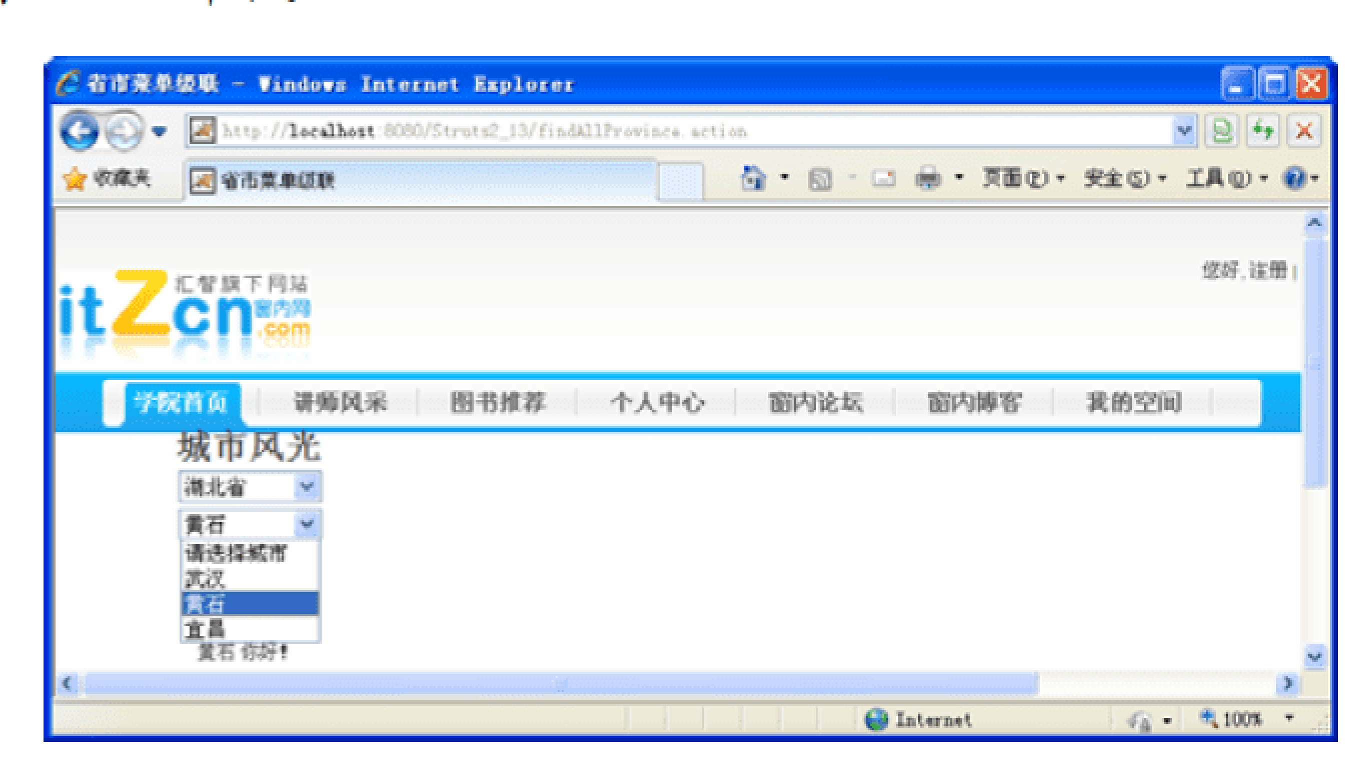


图 13-15　省市级联



第 14 章　太极研修院企业网站

内容摘要：

为了宣传一个太极研修院的实力，宣扬研修院的品牌、展示太极产品，以供对太极感兴趣的成员可以放心的选择本研修院进行学习，XX 太极研修院决定通过网络向广大群众展示出研修院的实力和魄力。本网站采用 Strut 2+Hibernate 3 两大框架实现了产品展示，并通过联系研修院的教员购买产品、查阅研修院的最新动态、向教员咨询问题、在线观看视频等功能。开发工具选择了 MyEclipse7.0、MySql5.0 和 JDK1.6。

学习目标：

- 掌握 Struts 2+Hiberante3 开发模式。
- 掌握 Struts 2、Hibernate 配置文件中的配置项。
- 熟练使用 Ant 技术生成 Hibernate 映射文件。
- 掌握 FCKEditor 与 JSP 整合的使用。
- 熟练的配置 Struts 2 中的配置文件。
- 掌握分页技术。

14.1　太极研修院企业网站简介

太极研修院是一个培训太极拳法的学院，因此它有自己学院的风采。它会在每年的不同时间招生，以供那些爱好习武的人学习太极拳法。在这个科技发展的时代，怎样把招生消息传播给天下所有爱好习武的人呢？当然是网络，因此太极研修院企业网站由此成立……

14.1.1　系统功能

本章所介绍的网站是一个功能有点复杂的太极研修院企业网站，本网站以展示为目的，向读者展示一种良好的程序架构。

1．前台功能

本网站前台主要分为首页展示、企业简介、新闻中心、太极商城、在线视频、太极风采、培训招生、太极感悟、个人博客、友情链接和联系我们 11 个栏目，前台功能模块图解如图 14-1 所示。

下面是各栏目功能模块的详解。

1)　首页展示

首页是一个网站的第一窗口，是浏览者对企业文化第一印象认知度的关键页面。页面的布局和页面风格的设定，对网站整体定位起着决定性的作用。在本栏目中简单的介绍太极研修院的概况及背景，以供浏览者对这个培训中心有个初步的认识。这里使用一个静态页面来展示。

2)　企业简介

向浏览者介绍太极研修院，管理员可以在后台管理系统中对其进行添加、修改、删除操作。企业简介的内容可以是图片、Flash、视频、文字等，这里使用 FCKeditor 框架与 JSP 整合实现。

3)　新闻中心

新闻中心存在二级栏目，即：太极动态、理论天地、太极养生、武林资讯和疑惑解答，对于前 4 个栏目下的新闻，浏览者可以查看新闻详情及对该条新闻的所有评论，还可以对该条新闻进行评论；对于疑惑解答栏目下的新闻(即浏览者提问的问题)，浏览者可以对其进行回答，对于别人的回答，浏览者还可以查看别人对该回答内容进行的评论，也可以对其进行评论。在显示新闻和显示评论时，都使用分页显示功能。

4)　太极商城

太极商城也存在二级栏目，即：音像制品、太极服装、太极器械和太极书籍。浏览者可以查看商城中的所有商品的详情，还可以购买(这里没有实现真正意义上的购买，只是单击“SHOP NOW”按钮时，跳转至联系我们页面)。

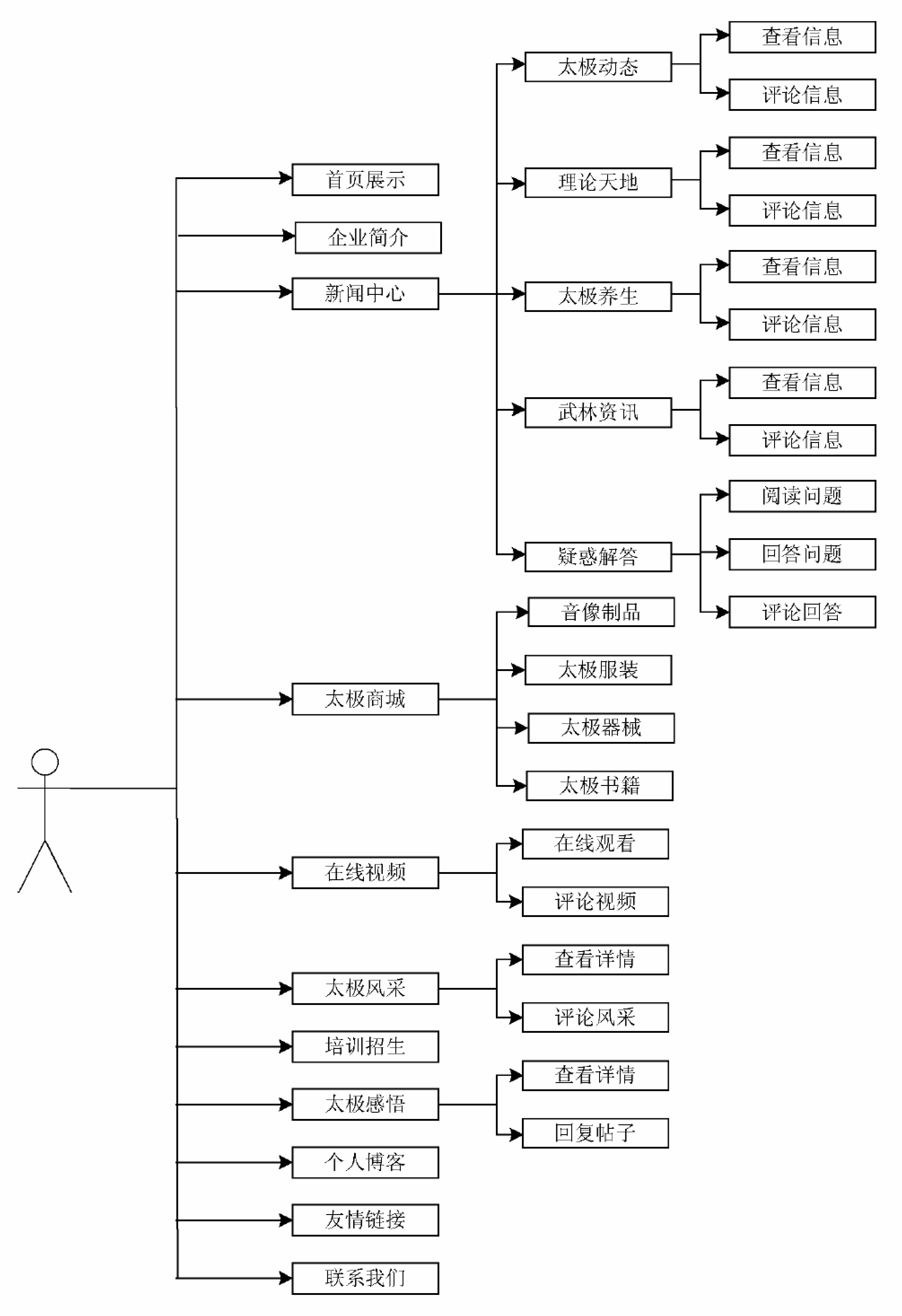


图 14-1　前台功能模块图解

5) 在线视频

浏览者可以查看所有有关太极研修院的视频内容，并对其进行评论。显示视频和显示评论都使用分页显示功能。

6) 太极风采

太极风采栏目向浏览者展示本研修院学徒的一些风采(比如在什么活动中获得了哪些奖项)，并以图片的列表形式在页面中分页显示，当单击某张图片时，显示该条信息的详细信息，浏览者可以对其进行评论。

7) 培训招生

这个栏目和企业简介模块功能的实现相同，只是向浏览者显示数据，管理员可以在后台对数据进行修改、删除、添加操作。

8) 太极感悟

太极感悟是管理员与学徒之间沟通的桥梁，管理员可以在后台发表学太极的一些心得，浏览者可以对其进行回复。当浏览者访问某条感悟信息一次，访问量多一(在 Web 开发中，很多时候要用到此功能，不可忽视)。

9) 个人博客

这只是一个超链接，链接至教员的博客。

10) 友情链接

对于一个成功的企业网站来说，这个栏目必不可少。通过本网站可以链接到其他网页(或网站)。

11) 联系我们

在太极商城栏目中已经提到过此栏目，以供浏览者联系我们。

2. 后台功能

本网站后台主要分为：新闻中心、太极商城、信息管理、用户管理、日志管理和系统信息 6 个栏目，下面是各功能模块详解。

1) 新闻中心

这个模块供管理员发表新闻、修改新闻、进行置顶操作、删除新闻，功能模块如图 14-2 所示。

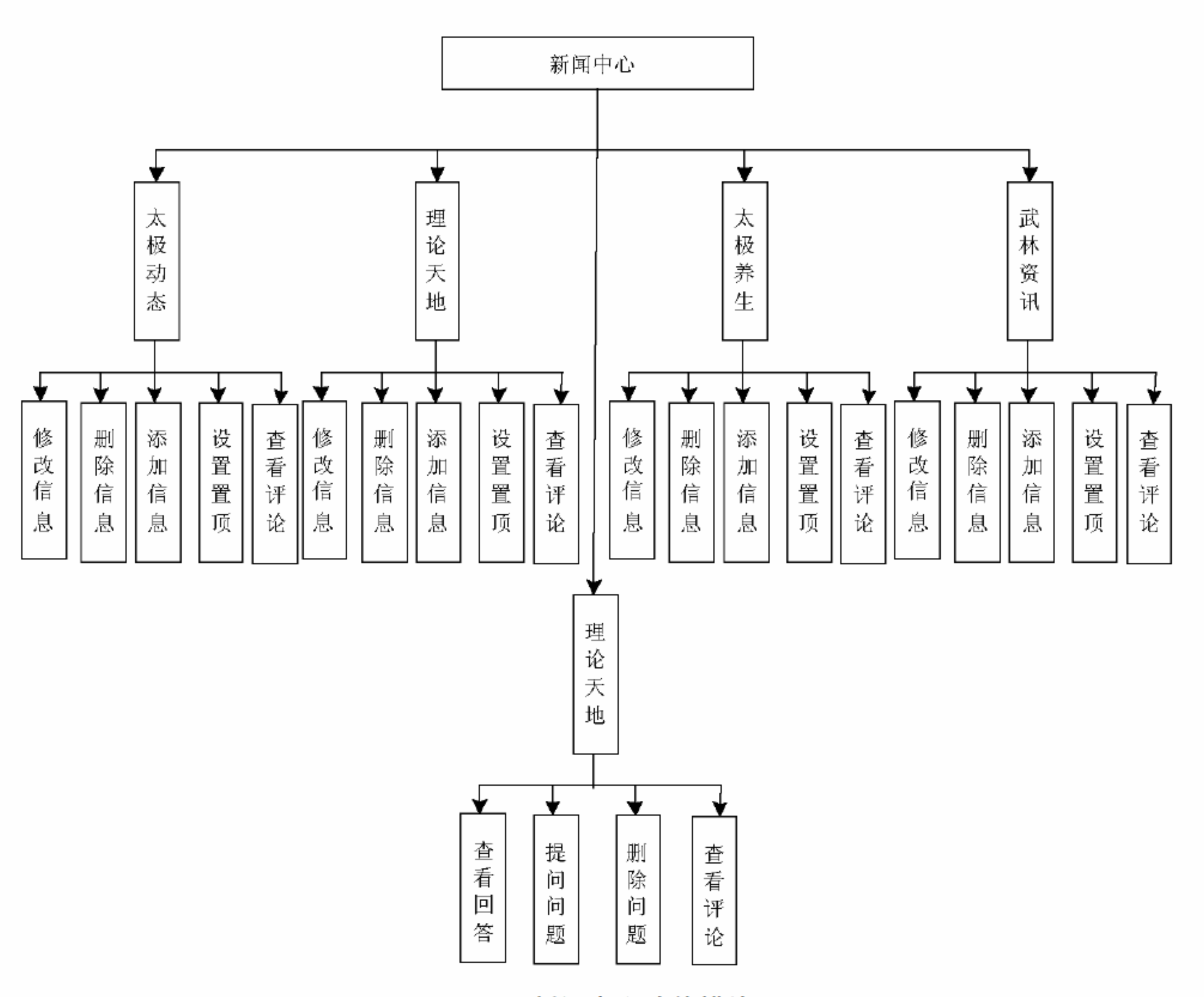


图 14-2　新闻中心功能模块图

2)　太极商城

这个模块供管理员添加商品、修改商品信息、删除商品信息，功能模块如图 14-3 所示。

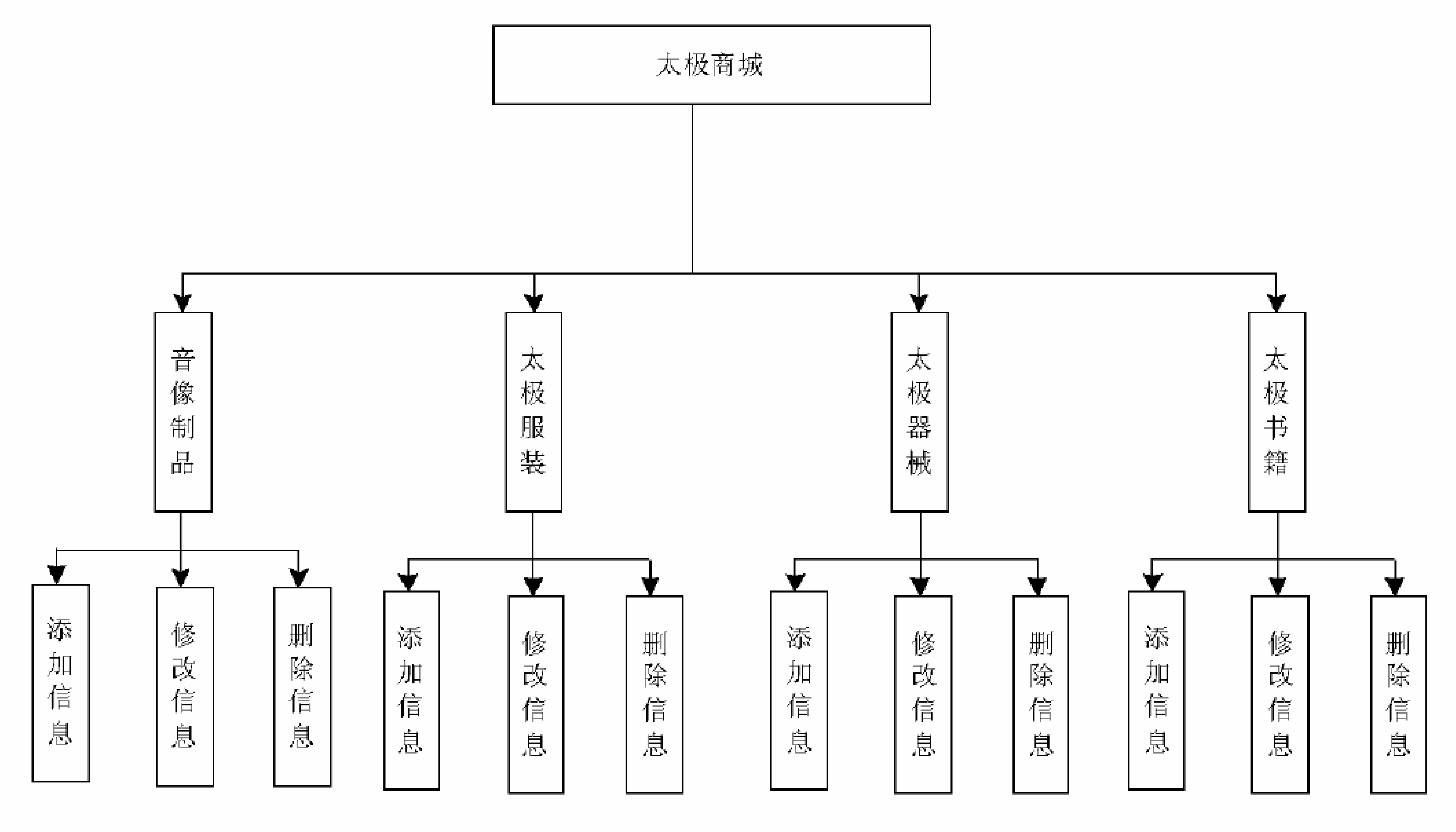


图 14-3　太极商城功能模块图

3) 信息管理

此模块是本系统的核心模块，前台中的大部分栏目功能的实现都包含在此功能模块中，其中包含企业简介、在线视频、联系我们、培训招生、友情链接、太极感悟、太极风采、评论信息和首页幻灯片的管理模块，管理员可以对这些信息进行添加、删除、修改操作，功能模块如图 14-4 所示。

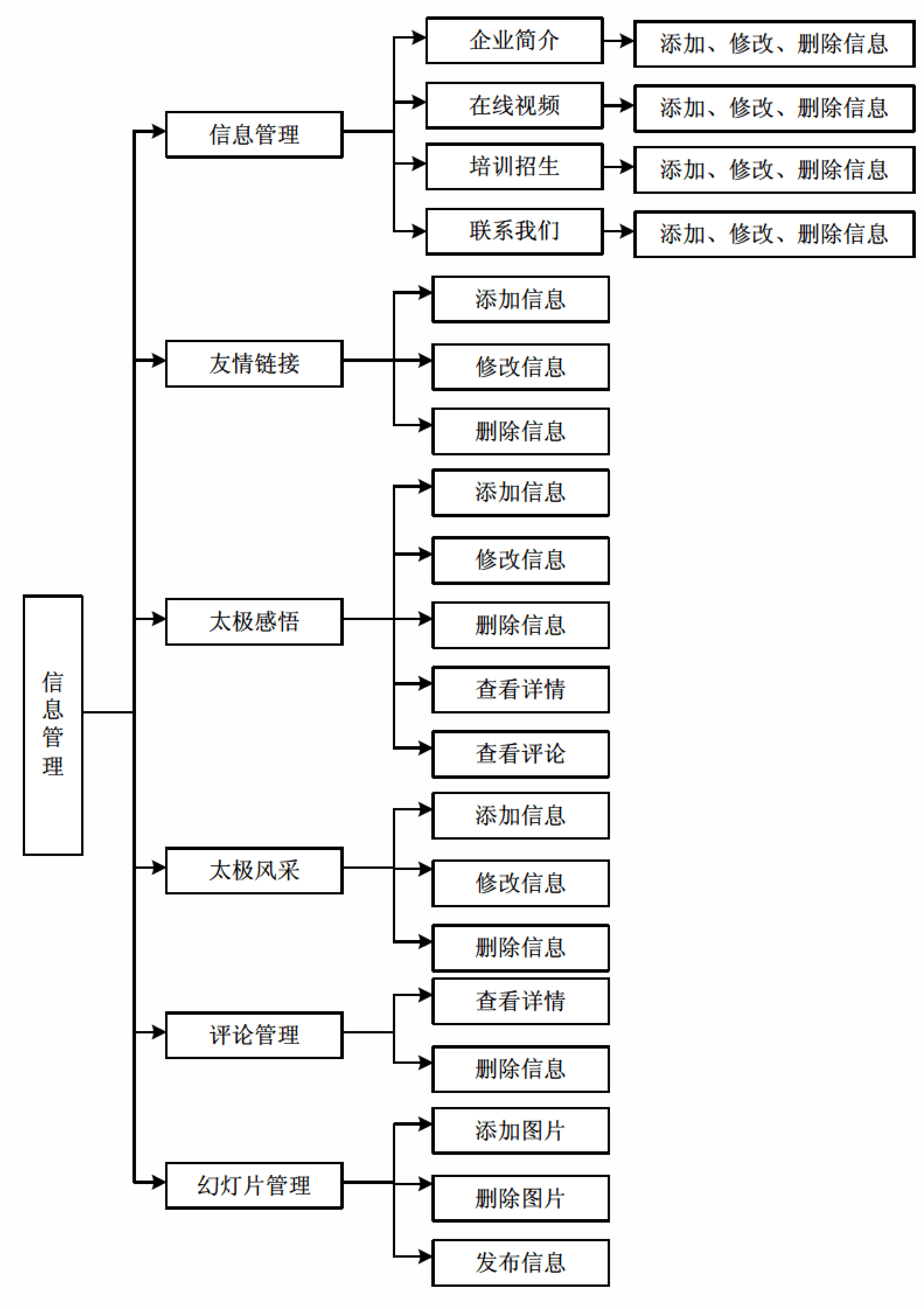


图 14-4 信息管理功能模块图

4) 用户管理

在此模块中，管理员可以查看所有的用户信息，包括管理员信息和会员信息，并对其进行添加、修改、删除等操作；管理员登录后台管理系统后可以修改自己的一些基本信息；管理员还可以对角色进行管理。用户管理栏目功能模块如图 14-5 所示。

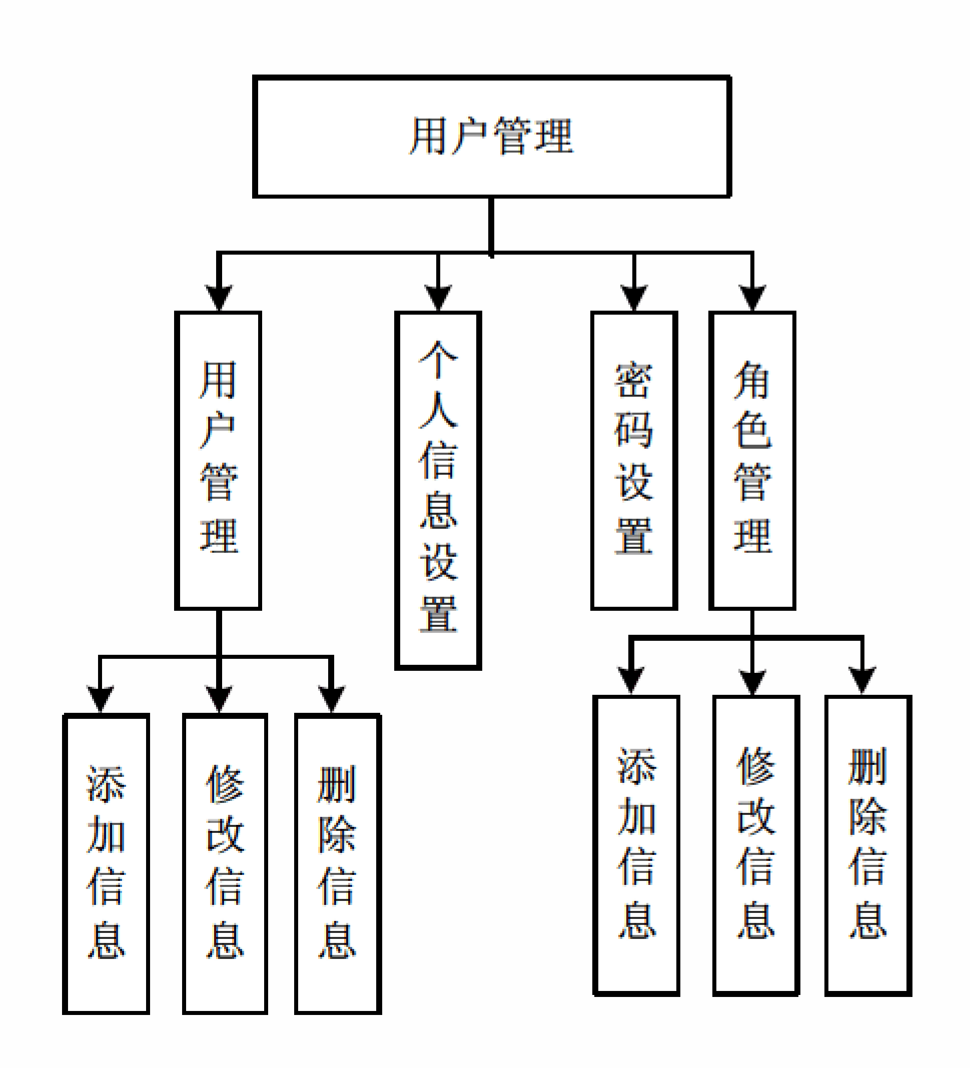


图 14-5　用户管理功能模块图

5)　日志管理

这个模块供管理员查看所有对本网站后台管理系统操作的信息，并对其进行一键删除。

6)　系统信息

此模块只是显示本系统的一些相关信息。

14.1.2　系统架构

本系统严格采用 Java EE 的三层机构，分为表现层、业务逻辑层和数据服务层。三层体系将业务规则、数据访问等工作放到中间层处理，客户端不直接与数据库交互，而是通过控制器与中间层建立连接，再有中间层与数据库交互。

表现层禁止 JSP 内嵌 Java 脚本，因而比较简单，JSP 页面使用 Struts 2 标签来实现数据，生成页面显示效果。中间层采用 Struts 2+Hibernate。

如果需要将 Struts 2 与 Hibernate 整合，必然有如图 14-6 所示的架构方案。

可能有读者对图 14-6 感到好奇：为什么还有 JDBC 技术呢？使用 Hibernate 框架不是可以代替 JDBC 技术吗？Hibernate 框架可以代替 JDBC 技术，一旦使用了 Hibernate 框架，无需在程序中使用 JDBC 技术，但是 Hibernate 的数据库访问是建立在 JDBC 技术基础之上的。这就是说，虽然使用 Hibernate 无需显式的使用 JDBC 编程，但 Hibernate 框架本身还必须借助于 JDBC 技术。

本项目的中间层组件可分为两个层。

- 业务逻辑层：该层的组件专注于业务逻辑的实现，避免与任何的持久化技术耦合。
- DAO 层：该层里包含大量的 DAO 组件，每个 DAO 组件专注于底层持久化实现，一个具体的 DAO 组件只能与特定的持久化技术耦合。

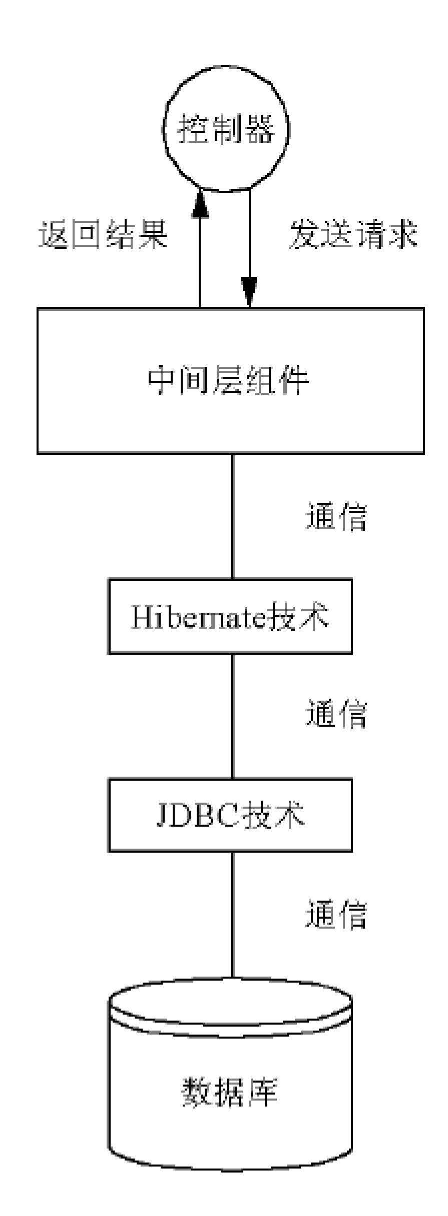


图 14-6　Struts 2 和Hiberante整合架构

14.2　数据库设计和实现

一个合理而适用的数据库表设计对于一个成功的 Web 开发来说占着主干的位置，这一节将为读者讲解一下太极研修院企业网站的数据库设计和实现。

太极研修院企业网站采用 MySql5.5 数据库。首先创建一个数据库：dwtj，接着为读者具体分析一下数据库表结构。

1．用户信息表(User)

用户信息表结构如表 14-1 所示。

表 14-1　用户信息表结构

字段名称	含　义	类　型	约　束
id	用户 Id	int	主键
username	用户名	varchar(255)	非空
password	密码	varchar(255)	非空
realname	真实姓名	varchar(255)	可以空
phone	联系电话	varchar(255)	可以空
address	联系地址	varchar(255)	可以空
e-mail	邮箱地址	varchart(255)	可以空
roleId	角色 Id，引用角色表(Role)中的 id 字段	int	非空

2．角色信息表(Role)

角色信息表结构如表 14-2 所示。

表 14-2　角色信息表结构

字段名称	含　义	类　型	约　束
id	角色 Id	int	主键
name	角色名称	varchar(255)	非空
tempString	备用字段	varchar(255)	可以空

3．新闻中心、企业简介、在线视频、联系我们、培训招生信息表(Messages)

因为这 5 个栏目的内容格式大同小异，因此新闻中心、企业简介、在线视频、联系我们、培训招生 5 大栏目的信息内容用一个数据表来实现，表结构如表 14-3 所示。

表 14-3　新闻中心、企业简介、在线视频、联系我们、培训招生信息表结构

字段名称	含　义	类　型	约　束
id	信息 Id	int	主键
name	信息标题	varchar(255)	非空
subject	信息内容链接路径	varchar(255)	非空
createTime	创建时间	date	非空
langType	语言类型，分为中、英文两种类型	varchar(255)	非空
msyType	信息类型，这个表中包含有新闻中心、企业简介、在线视频、联系我们、培训招生等栏目的信息，这个字段标识特定的栏目	int	非空
tempStr	信息来源	varchar(255)	可以空
createUser	创建人	varchar(255)	非空
headMsg	文章导读	varchar(255)	非空
clickNum	点击率	int	非空
parentId	上级栏目 Id，引用信息表(Messages)中的 id 字段	int	非空
sign	是否置顶，1 表示置顶，0 表示默认	int	非空

4．商品信息表(Products)

商品信息表结构如表 14-4 所示。

表 14-4　商品信息表结构

字段名称	含　义	类　型	约　束
id	商品 Id	int	主键
name	商品名称	varchar(255)	非空
intro	商品简介	text	非空
price	商品价格	double	非空
number	商品编号	varchar(255)	非空
pic	商品图片路径	varchar(255)	非空
langType	语言类型	varchar(255)	非空
tempString	备用字段	varchar(255)	可以空
msgType	商品信息类型，二级栏目标识	int	非空

5. 问题信息表(Questions)

新闻中心的二级栏目中存在一个疑惑解答栏目，它与其他 4 个栏目的信息结构不同，这里把它抽取出来使用一张表来实现，问题信息表结构如表 14-5 所示。

表 14-5 问题信息表结构

字段名称	含 义	类 型	约 束
id	问题 Id	int	主键
question	问题内容	varchar(255)	非空
createTime	提问时间	date	非空
userId	提问者，引用用户表(User)中的 id 字段	int	非空
tempString	备用字段	varchar(255)	可以空

6. 回答问题信息表(Answers)

上面是问题表的结构设计，这里所要讲的就是针对上面问题的回答内容的信息表设计，表结构如表 14-6 所示。

表 14-6 回答问题信息表结构

字段名称	含 义	类 型	约 束
id	回答信息 Id	int	主键
createUser	回答者姓名	varchar(255)	非空
sourceUrl	来源	varchar(255)	非空
content	回答内容	varchar(255)	非空
clickNum	点击率	int	非空
questioned	问题 Id，引用提问问题表(Questions)表中的 id 字段	int	非空
createTime	回答时间	date	非空
hostNum	支持人数	int	非空
tempString	备用字段	varchar(255)	可以空

7. 太极风采信息表(Student)

太极风采信息表结构如表 14-7 所示。

表 14-7 太极风采信息表结构

字段名称	含 义	类 型	约 束
id	信息 Id	int	主键
name	信息名称	varchar(255)	非空
createTime	创建时间	date	非空
imageUrl	图片链接地址	varchar(255)	非空
langType	语言类型	varchart(255)	非空
content	信息内容	text	非空
title	信息标题	varchar(255)	非空
tempString	备用字段	varchar(255)	可以空

8．太极感悟信息表(PrenticeMsg)

太极感悟信息表结构如表 14-8 所示。

表 14-8　太极感悟信息表结构

字段名称	含　义	类　型	约　束
id	信息 Id	int	主键
name	信息名称	varchar(255)	非空
createTime	创建时间	date	非空
content	信息内容	varchar(255)	非空
title	信息标题	varchar(255)	非空
checkedId	点击率	int	非空
tempString	备用字段	varchar(255)	可以空
langType	语言类型	varchar(255)	非空

9．友情链接信息表(FriendLink)

友情链接信息表结构如表 14-9 所示。

表 14-9　友情链接信息表结构

字段名称	含　义	类　型	约　束
id	信息 Id	int	主键
title	友情链接标题	varchar(255)	非空
addressUrl	链接地址	varchar(255)	非空
imgUrl	显示图片链接路径	varchar(255)	非空
tempString	备用字段	varchar(255)	可以空

10．评论信息表(ReplyMsg)

新闻、太极感悟、疑惑解答中的回答内容都存在评论功能，因此评论信息表必不可少，表结构如表 14-10 所示。

表 14-10　评论信息表结构

字段名称	含　义	类　型	约　束
id	信息 Id	int	主键
content	评论内容	varchar(255)	非空
createUser	评论人姓名	varchar(255)	非空
createTime	评论时间	date	非空
topicId	主贴 Id，引用信息表(Messages)、太极感悟表(PrenticeMsg)、回答问题表(Answers)表中的 id 字段	int	非空
topicType	主贴类型。1 表示新闻，2 表示太极感悟，3 表示回答问题	int	非空
parented	对评论进行评论，引用评论信息表(ReplyMsg)表中的 id 字段	int	非空
tempString	备用字段	varchar(255)	可以空

11. 系统日志表(SystemLogs)

系统日志信息表结构如表 14-11 所示。

表 14-11　系统日志信息表结构

字段名称	含　义	类　型	约　束
id	日志信息 Id	int	主键
username	操作者	varchar(255)	非空
loginTime	操作时间	date	非空
loginIP	操作者的 IP 地址	varchar(255)	非空
operationName	操作内容	varchar(255)	非空
userId	操作者 Id，引用用户表(User)中的 id 字段	int	非空
tempStr	备用字段	varchar(255)	可以空

14.3　后台模块——新闻中心

由于此项目功能的实现太多，我不再一一介绍，读者可以根据本章源码来做具体的分析，这里只讲解典型的模块功能的实现。

此项目在 web.xml 文件中，配置的 Struts 2 控制器(org.apache.struts2.dispatcher.FilterDispatcher)的拦截路径为“.action”。

14.3.1　查询新闻信息，分页显示

前面已经提到新闻中心栏目中包括太极动态、理论天地、太极养生、武林资讯和疑惑解答 5 个栏目，而前 4 个栏目信息存储在同一张表中(Messages 表)，在 Messages 表中有一个 msgType 字段，定义太极动态的 msgType 值为 7、理论天地的 msgType 值为 8、太极养生的 msgType 值为 9、武林资讯的 msyType 值为 10。下面来具体的讲解一下实现步骤。

(1) 在项目中新建 com.dwtj.model 包，在该包下新建 Messages 类。由实体类生成映射文件，采用 Ant 技术，在实体类中的属性上配置如下代码格式。Messages 类的内容如下。

```
public class Messages {
    /**
     * @hibernate.id
     * generator-class="native"
     */
    private int id;//信息 Id，主键
    /**
     * @hibernate.property
     */
    private String name;// 消息名称
```

```
    /**
     * @hibernate.property
     */
    private String subject;// 映射路径
    /**
     * @hibernate.property
     */
    private Date createTime;// 创建时间
    /**
     * @hibernate.property
     */
    private String langType;// 语言类型
    /**
     * @hibernate.property
     */
    private int msgType;// 消息类别
    /**
     * @hibernate.property
     */
    private String createUser;//创建人
    /**
     * @hibernate.property
     */
    private String headMsg;//文章导读
    /**
     * @hibernate.property
     */
    private String tempStr;// 信息来源
    /**
     * @hibernate.property
     */
    private int clickNum;//点击量
    /**
     * 上级菜单
     * @hibernate.property
     */
    private int parentId;
    /**
     * 是否置顶
     * @hibernate.property
     */
    private int sign;
    /*下面是上面所有属性的 set、get 方法，这里省略*/
}
```

(2) 新闻栏目中的所有信息都要用到分页显示，因此必须要为分页做好准备。新建 com.dwtj.common 包，在该包下新建 PageBean 类，内容如下。

```
public class PageBean {
    private String hql = ""; // 查询的语句
    private int nowpage = 0; // 当前页
    private Integer totalpage = 0; // 总页数
    private int pagesize = 20; // 每页显示最大值
    private int totalCount =0; //当前查询的总条数
    @SuppressWarnings("unchecked")
    private List result = new ArrayList(0);
    /*下面是上面所有属性的 set、get 方法，这里省略*/
}
```

(3) 在 com.dwtj.common 包下新建分页处理类 PageCtr，内容如下。

```
public class PageCtr {
    public static PageBean  PageCtrUpdate(PageBean pageBean ,String mark,int
number){
        //如果 mark 为“f”，则跳转至首页
        if (mark.equals("f")) {
            pageBean.setNowpage(0);

        }
        //如果 mark 为“l”，则跳转至尾页
        else if (mark.equals("l")) {
            pageBean.setNowpage(pageBean.getTotalpage() - 1);
        }
        //如果 mark 为“p”，则跳转至上一页
        else if (mark.equals("p") && pageBean.getNowpage() != 0) {
            pageBean.setNowpage(pageBean.getNowpage() - 1);
        }
        //如果 mark 为“n”，则跳转至下一页
        else if (mark.equals("n")
                && pageBean.getNowpage() != pageBean.getTotalpage() - 1) {
            pageBean.setNowpage(pageBean.getNowpage() + 1);
        }
        //如果 mark 为“t”，则跳转至指定页码
        else if(mark.equals("t")){
            pageBean.setNowpage(number-1);
        }
        return pageBean;
    }
}
```

(4) 创建 DAO 层架构。在项目中新建 com.dwtj.dao 包，该包下存放了持久层的所有接口，在该包下新建 MessagesDao 接口，负责与持久化对象交互，封装了数据的增、删、改、查等操作，内容如下。

```
public interface MessagesDao {
    //添加信息
    public boolean addMessage(Messages msg);
    //修改信息
    public boolean updateMessage(Messages msg);
```

```
    //删除信息
    public boolean deleteMessage(int msgId);
    //分页查询信息
    public PageBean selectMessagesNewsAll(PageBean pageBean, String sql)
}
```

(5)　创建 DAO 层实现类。在项目中新建 com.dwtj.dao.impl 包，在该包下新建 MessagesDao 接口的实现类 MessageDaoImpl，实现 MessagesDao 接口中的方法，内容如下。

```
public class MessageDaoImpl implements MessagesDao {
    //添加信息
    public boolean addMessage(Messages msg) {
        //定义返回结果
        boolean con=false;
        //定义 Session 对象
        Session session=null;
        //定义 Transaction 对象
        Transaction tx=null;
        try{
            //获取 Session 对象，调用 HibernateSessionFactory 类中的 getSession()
              方法
            session=HibernateSessionFactory.getSession();
            //获取 Transaction 对象，调用 Session 中的 beginTransaction()方法
            tx=session.beginTransaction();
            //调用 Session 中的 sava(entity entity)方法保存数据
            session.save(msg);
            con=true;
            //提交事务
            tx.commit();
        }catch(Exception ex){
            ex.printStackTrace();
            tx.rollback();
        }finally{
            session.close();
        }
        return con;
    }
    //删除信息
    public boolean deleteMessage(int msgId) {
        boolean con=false;
        Session session=null;
        Transaction tx=null;
        try{
            //获取 Session 对象
            session=HibernateSessionFactory.getSession();
            //获取 Transaction 对象
            tx=session.beginTransaction();
            //获取特定的信息
            Messages msg=(Messages)session.get(Messages.class, msgId);
            //这里使用 FCKeditor 与 JSP 整合，因此需要把 FCKeditor 内容页面从存放路径中
              删除掉
```

```
            String pathDb =
ServletActionContext.getServletContext().getRealPath(
                    msg.getSubject());
            ReadInFile.judgefilename(pathDb);
            //删除数据
            session.delete(msg);//删除信息
            con=true;
            tx.commit();
        }catch(Exception ex){
            ex.printStackTrace();
            tx.rollback();
        }finally{
            session.close();
        }
        return con;
    }
    //修改信息
    public boolean updateMessage(Messages msg) {
        Session session=null;
        Transaction tx=null;
        boolean con=false;
        try{
            //获取 Session 对象
            session=HibernateSessionFactory.getSession();
            //获取 Transaction 对象
            tx=session.beginTransaction();
            //把修改后的值重新覆盖修改前的值
            Messages messages=(Messages)session.get(Messages.class,
msg.getId());
            messages.setCreateTime(msg.getCreateTime());
            messages.setCreateUser(msg.getCreateUser());
            messages.setHeadMsg(msg.getHeadMsg());
            messages.setMsgType(msg.getMsgType());
            messages.setName(msg.getName());
            messages.setSubject(msg.getSubject());
            messages.setClickNum(msg.getClickNum());
            messages.setTempStr(msg.getTempStr());
            messages.setSign(msg.getSign());
            //修改数据
            session.update(messages);
            con=true;
            tx.commit();
        }catch(Exception ex){
            ex.printStackTrace();
            tx.rollback();
        }finally{
            session.close();
        }
        return con;
    }
    //根据不同的 sql 语句查询不同的信息
```

```
    public PageBean selectMessagesNewsAll(PageBean pageBean, String sql) {
        //获取 Session 对象
        Session session = HibernateSessionFactory.getSession();
        //根据不同的条件语句查询所有的信息，并分页显示
        List<Messages> list = session.createQuery(
                "from Messages "+sql+"  order by sign desc,id desc ").
                 setMaxResults(
                pageBean.getPagesize()).setFirstResult(
                pageBean.getNowpage() * pageBean.getPagesize()).list();
        pageBean.setResult(list);
        //获取符合条件的数据条数
        Object obj = session.createQuery(
                "select count(*) from Messages "+sql
                ).uniqueResult();
        int count = Integer.parseInt(obj.toString());
        //给 PageBean 类中的 totalCount 属性赋值
        pageBean.setTotalCount(count);
        //获取总页数
        count = (count + pageBean.getPagesize() - 1) / pageBean.getPagesize();
        pageBean.setTotalpage(count);
        return pageBean;
    }
}
```

(6) 创建业务层接口。在项目中新建 com.dwtj.service 包，在该包下新建 MessagesService 接口，编写需要实现的方法，代码如下。

```
public interface MessagesService {
    //添加信息
    public boolean addMessage(Messages msg);
    //修改信息
    public boolean updateMessage(Messages msg);
    //删除信息
    public boolean deleteMessage(int msgId);
    //根据不同的 sql 语句查询不同的信息
    public PageBean selectMessagesNewsAll(PageBean pageBean, String sql) ;
}
```

(7) 编写业务层实现类 MessageServiceImpl，实现上面接口中的 4 个方法，代码如下。

```
public class MessageServiceImpl implements MessagesService {
    MessagesDao msgDao=new MessageDaoImpl();
    //添加信息页面
    public boolean addMessage(Messages msg) {
        return msgDao.addMessage(msg);
    }
    //删除信息
    public boolean deleteMessage(int msgId) {
        return msgDao.deleteMessage(msgId);
    }
    //修改信息
```

```
    public boolean updateMessage(Messages msg) {
        return msgDao.updateMessage(msg);
    }
    //根据不同的条件查询所有信息
    public PageBean selectMessagesNewsAll(PageBean pageBean, String sql) {
        return msgDao.selectMessagesNewsAll(pageBean, sql);
    }
}
```

(8) 当单击后台管理系统左边菜单导航中新闻中心栏目下的二级栏目时，需要根据不同的 msgType 值来查询不同的新闻信息数据。在 com.dwtj.actions 包下新建 MessageAction 类，继承自 com.opensymphony.xwork2.ActionSupport 类，编写分页显示信息数据方法，代码如下。

```
    //引用 Messages 实体类
    private Messages msg;
    /*省略 Messages 对象属性 msg 的 set、get 方法*/

    //查询新闻中心二级菜单中的内容
    public String findNews(){
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        //从 Session 对象中获取 PageBean 对象
        PageBean pageBean = (PageBean) request.getSession().getAttribute(
                "pageBean");
        if (pageBean == null) {
            pageBean = new PageBean();
        }
        String mark = "";
        //如果从页面中传过来的 mark 值不为 null，获取 mark 值，否则跳转至首页
        if (request.getParameter("mark") != null) {
            mark = request.getParameter("mark");
        } else {
            mark = "f";
        }
        //定义要跳转的页码
        int tempNumber = 0;
        if (request.getParameter("number") != null) {
            tempNumber = Integer
                    .parseInt(request.getParameter("number").trim());
        }
        // 调用业务层分页显示数据
        pageBean = msgService.selectMessagesNewsAll(PageCtr.PageCtrUpdate(
                pageBean, mark, tempNumber), " where
msgType="+msg.getMsgType());
        request.setAttribute("msgs", pageBean.getResult());
        request.setAttribute("msgType", msg.getMsgType());
        pageBean.setResult(null);
        request.getSession().setAttribute("pageBean", pageBean);
        return "secondMsg";
    }
```

(9) 在项目的根目录下新建 Ant 的辅助文件 build.xml，使用它可以将实体类 Messages 自动生成 Messages.hbm.xml 映射文件和 hibernate.cfg.xml 配置文件。内容如下。

```
<?xml version="1.0" encoding="GBK"?>
<project name="太极研修院构建脚本" default="生成 Hibernate 配置文件" basedir=".">
    <property name="src.dir" value="${basedir}/src"/>
    <property name="build.dir" value="${basedir}/bin"/>
    <property name="xdoclet.home" value="E:\soft\xdoclet-plugins-1.0.3"/>
    <!-- Build classpath -->
    <path id="xdoclet.task.classpath">
        <fileset dir="${xdoclet.home}/lib">
            <include name="**/*.jar"/>
        </fileset>
        <fileset dir="${xdoclet.home}/plugins">
            <include name="**/*.jar"/>
        </fileset>
    </path>
    <taskdef
        name="xdoclet"
        classname="org.xdoclet.ant.XDocletTask"
        classpathref="xdoclet.task.classpath"
    />
    <target name="生成 Hibernate 配置文件">
        <xdoclet>
            <fileset dir="${src.dir}/com/dwtj/model">
                <include name="**/*.java"/>
            </fileset>
            <component

    classname="org.xdoclet.plugin.hibernate.HibernateConfigPlugin"
                destdir="${src.dir}"
                version="3.0"
                hbm2ddlauto="update"
                jdbcurl="jdbc:mysql://localhost:3306/dwtj"
                jdbcdriver="com.mysql.jdbc.Driver"
                jdbcusername="root"
                jdbcpassword="000"
                dialect="org.hibernate.dialect.MySQLDialect"
                showsql="true"
            />
        </xdoclet>
    </target>
    <target name="生成 hibernate 映射文件">
        <xdoclet>
            <fileset dir="${src.dir}/com/dwtj/model">
                <include name="**/*.java"/>
            </fileset>
            <component

    classname="org.xdoclet.plugin.hibernate.HibernateMappingPlugin"
                version="3.0"
```

```
                destdir="${src.dir}"
            />
        </xdoclet>
    </target>
</project>
```

(10) 选择 MyEclipse 中的 Window | Show View | Other… | Ant | Ant 选项，打开 Ant 窗口。单击窗口工具栏中的图标，添加上面编写好的 build.xml 文件，出现如图 14-7 所示的结构，右击“生成 hibernate 映射文件”，选择 Run As | Ant Build，生成 Messages 映射文件。生成 Hibernate 配置文件和生成 Hiberante 映射文件步骤一样。

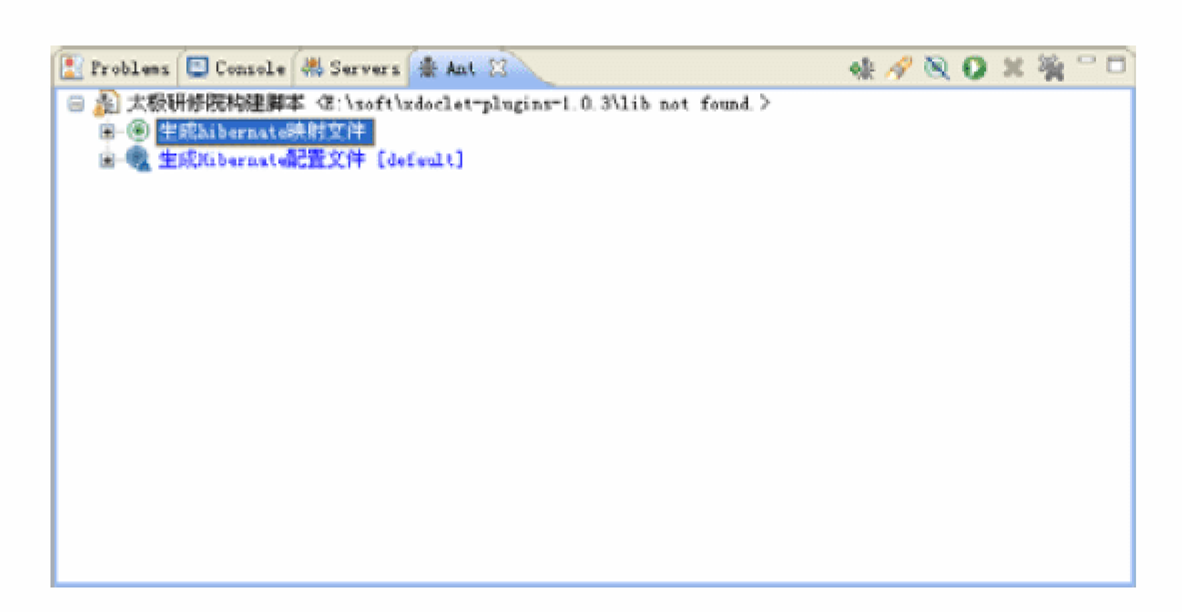


图 14-7　Ant结构

(11) 在 src 下新建 struts.xml 文件，配置异常处理。struts.xml 文件内容如下。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<constant name="struts.i18n.encoding" value="gbk" />
    <constant name="struts.devMode" value="true" />
<package name="ma" namespace="/" extends="struts-default">
        <!-- 自定义拦截器用于检测是否登录 -->
        <global-results>
            <result name="error">/common/exception.jsp</result>
        </global-results>
        <global-exception-mappings>
            <exception-mapping result="error"
                exception="java.lang.RuntimeException">
            </exception-mapping>
        </global-exception-mappings>
    </package>
</struts>
```

(12) 新建 struts-config 文件，新建 message.xml 文件，配置 MessageAction 类，配置如下。

```
    <package name="message" namespace="/message" extends="ma">
        <action name="message" class="com.dwtj.actions.MessageAction">
            <result name="secondMsg">/admin/news/index.jsp</result>
        </action>
    </package>
```

(13) 在 admin 文件夹下的 left.jsp 页面中添加导航菜单，内容如下。

```
<TR>
        <TD width="2%"><IMG src="admin/Images/closed.gif"></TD>
        <TD height=23><A
href="message/message!findNews.action?msg.msgType=7" target=main>太极动态
</A></TD>
</TR>
<TR>
        <TD><IMG src="admin/Images/closed.gif"></TD>
        <TD height=23><A
href="message/message!findNews.action?msg.msgType=8" target=main>理论天地
</A></TD>
</TR>
<TR>
        <TD><IMG src="admin/Images/closed.gif"></TD>
        <TD height=23><A
href="message/message!findNews.action?msg.msgType=9"
target=main>太极养生</A> </TD>
</TR>
<TR>
        <TD><IMG src="admin/Images/closed.gif"></TD>
        <TD height=23><A
href="message/message!findNews.action?msg.msgType=10"
target=main>武林资讯</A> </TD>
</TR>
```

(14) 在 admin 下新建 news 文件夹，在 news 文件夹下新建 index.jsp 页面，分页显示查询出来的数据，代码如下。

```
<%@ page language="java" import="java.util.*" pageEncoding="gbk"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<table class="table" cellspacing="1" cellpadding="2" width="100%"
align="center" border="0">
    <tbody>
        <tr>
            <th height="25" colspan="5" align="left" class="man">
                <s:if test="#request.msgType==7">太极动态</s:if>
                <s:elseif test="#request.msgType==8">理论天地</s:elseif>
                <s:elseif test="#request.msgType==9">太极养生</s:elseif>
                <s:elseif test="#request.msgType==10">武林资讯</s:elseif>
            </th>
            <th height="25" align="right" class="addMsg" colspan="4">添加新闻
 发布最新新闻
            </th>
        </tr>
        <tr>
            <td class="td_bg" width="15%" height="25">
                <div align="center">信息标题</div>
            </td>
            <td class="td_bg" width="10%">
```

```
                <div align="center">创建时间</div>
            </td>
            <td width="10%" class="td_bg">
                <div align="center">创建人</div>
            </td>
            <td width="15%" class="td bg">
                <div align="center">文章导读</div>
            </td>
            <td width="5%" class="td_bg">
                <div align="center">语言</div>
            </td>
            <td width="10%" class="td_bg">
                <div align="center">评论</div>
            </td>
            <td class="td_bg" width="6%">
                <div align="center">置顶</div>
            </td>
            <td class="td bg" width="6%">
                <div align="center">编辑</div>
            </td>
            <td class="td bg" width="6%">
                <div align="center">删除</div>
            </td>
        </tr>
    <s:if test="#request.msgs.size()!=0">
        <s:iterator value="#request.msgs" var="msg">
            <tr>
                <td class="td_bg"  height="25" style="color: #993333;">
                    <s:if test="#msg.sign==1">
                        <font style="font-size:12px;color:red;">顶</font>
                    </s:if>
                    <s:if test="#msg.name.trim().length()>30">
                        <s:property value="%{#msg.name.substring(0,30)}"/>
                    </s:if>
                    <s:else>
                        <s:property value="#msg.name"/>
                    </s:else>
                </td>
                <td class="td_bg">
                    <s:date name="#msg.createTime" format="yyyy-MM-dd"/>
                </td>
                <td class="td_bg">
                    <s:property value="#msg.createUser"/>
                </td>
                <td class="td bg">
                    <s:if test="#msg.headMsg.trim().length()>20">
                        <s:property value="%{#msg.headMsg.substring(0,20)}" />
                    </s:if>
                    <s:else>
                        <s:property value="#msg.headMsg"/>
                    </s:else>
```

```
                <td class="td_bg">
                    <s:if test="#msg.langType.equals(\"zh\")">中文</s:if>
                    <s:else>英文</s:else>
                </td>
                <td class="td bg" width="10%">【查看评论】</td>
                <td class="td bg" width="10%">
                    <s:if test="#msg.sign==0">【置顶】</s:if>
                    <s:else>【取消置顶】</s:else>
                </td>
                <td class="td_bg" width="10%">【编辑】</td>
                <td class="td_bg" width="10%">【删除】</td>
            </tr>
        </s:iterator>
    </s:if>
    <s:else>
        <tr>
            <td colspan="9" align="center" class="td bg" height="25px">暂无数
据</td>
        </tr>
    </s:else>
    <s:if test="#request.msgs.size()!=0">
        <tr>
            <th class="bg_tr" align="right" colspan="9" height="25">
                <s:push value="#request.session.pageBean">
                    共 <span class="red"><s:property value="totalCount"
/></span>条记录                 当前第<span class="red"><s:property
value="nowpage+1" /></span>/<s:property value="totalpage" />页 每页<s:property
value="pagesize" />条数据
                </s:push>
                <s:a
href="message/message!findNews.action?msg.msgType=%{#request.msgType}&mark=
f&number=0">首页</s:a>
                <s:a
href="message/message!findNews.action?msg.msgType=%{#request.msgType}&mark=
p&number=0">上一页</s:a>
                <s:a
href="message/message!findNews.action?msg.msgType=%{#request.msgType}&mark=
n&number=0">下一页</s:a>

                <s:a
href="message/message!findNews.action?msg.msgType=%{#request.msgType}&mark=
l&number=0">尾页</s:a>
            </th>
        </tr>
    </s:if>
    </tbody>
</table>
```

到此，分页显示新闻信息已经完成，单击网站后台管理系统左边导航菜单中的太极动态模块，出现如图 14-8 所示的页面。

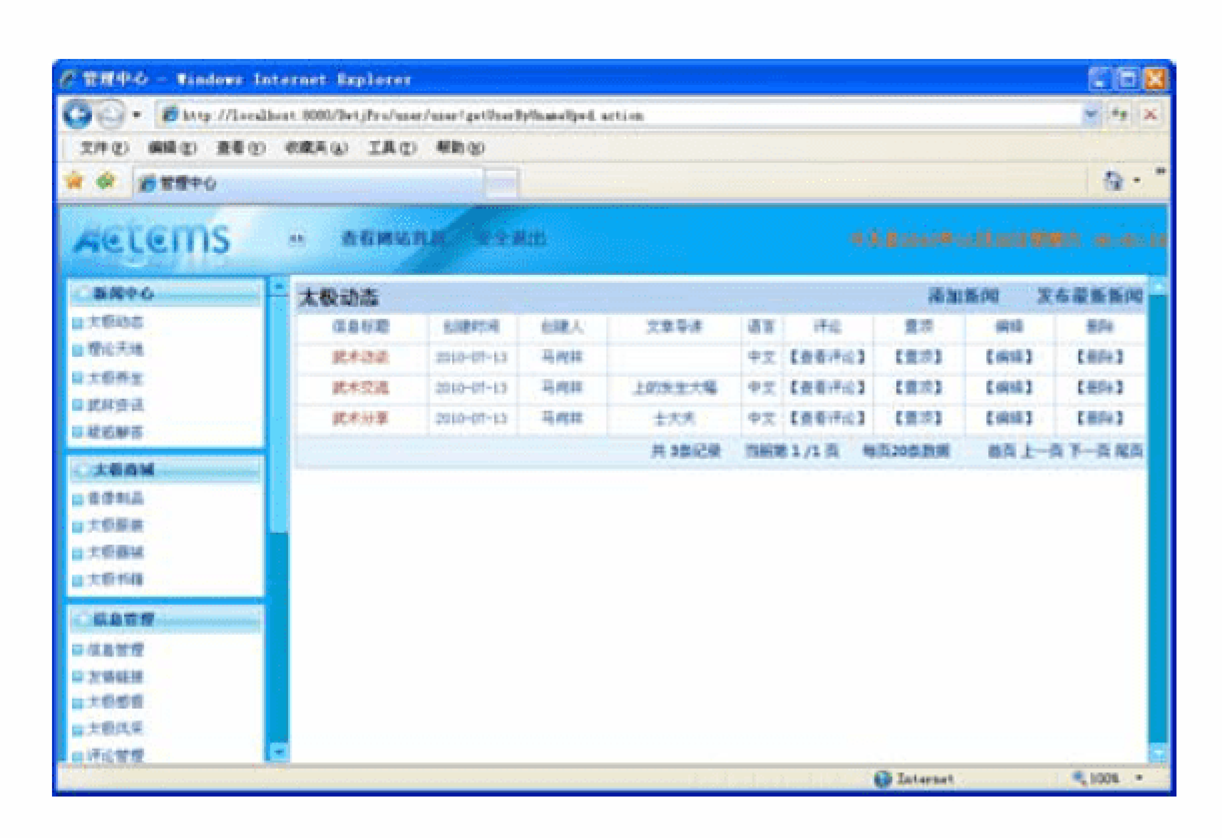


图 14-8　太极动态信息列表

当前显示的是第一页数据，每页显示 20 条，单击界面中的“下一页”按钮，页面跳转至第二页，但是数据少于 20 条，单击“下一页”按钮时，页面不会跳转。太极动态、理论天地、太极养生和武林资讯 4 个模块呈现出来的页面与上图 14-8 大同小异，只是数据不一样而已。

14.3.2　添加新闻信息

管理员可以在后台管理系统中添加新闻信息。

(1) 在 MessageAction 类中添加方法，实现新闻信息的添加功能，代码如下。

```
    //添加新闻
    public String addInputNews(){
        HttpServletRequest request=ServletActionContext.getRequest();
        //把 Messages 类中 msgType 属性值存放在 HttpServletRequest 对象中
        request.setAttribute("msgtype", msg.getMsgType());

        return "addNewsInput";
    }
```

(2) 在 struts-config 文件夹下的 message.xml 文件中配置“addNewsInput”的结果页面，配置代码如下。

```
<result name="addNewsInput">/admin/news/addInput.jsp</result>
```

(3) 在页面中添加 JavaScript 代码，在新窗口中打开添加新闻界面，代码如下。

```
function openWin(f, n, w, h, s) {
    sb = s == "1" ? "1" : "0";
    l = (screen.width - w) / 2;
    t = (screen.height - h) / 2;
    sFeatures = "left=" + l + ",top=" + t + ",height=" + h + ",width=" + w +
",center=1,scrollbars=" + sb + ",status=0,directories=0,channelmode=0";
    openwin = window.open(f, n, sFeatures);
    if (!openwin.opener) {
        openwin.opener = self;
    }
```

```
    openwin.focus();
    return openwin;
}
```

(4) 给页面中的“添加新闻”按钮中添加事件，代码如下。

```
<a
onclick="openWin('message/message!addInputNews.action?msg.msgType=<s:proper
ty value="#request.msgType"/>','addInput',900,700,1);">添加新闻</a>
```

(5) 当单击界面中的“添加新闻”按钮时，在新窗口中打开添加新闻界面，输入新闻的一些信息，单击添加新闻界面中的“保存信息”提交按钮，提交表单至 MessageAction 类中的 addMessage()方法(表单 action="message/message!addMessages.action")，执行添加新闻操作。MessageAction 类中的 addMessage()方法代码如下。

```
    //添加信息
    public String addMessages() {
        try {
            //获取 HttpServletRequest 对象
            HttpServletRequest request = ServletActionContext.getRequest();
            //获取登录用户信息，当用户登录时，检查用户是否合法用户，如果是，存储至
HttpServletRequest 对象中
            Users loginUser = (Users)
request.getSession().getAttribute("login");
            //新闻内容采用了 FCKEdirtor 与 JSP 整合，这里把 FCKeditor 中的内容生成了一
个页面,uploadDir 为页面存储路径
            String path =
ServletActionContext.getServletContext().getRealPath(
                    uploadDir);
            String pathDb = ServletActionContext.getServletContext()

    .getRealPath(RandomStringtable.genRandomNum("upLoadDB/"));
            //把添加的内容生成一个页面，并存放至指定的路径下
            LoadImgforString lifs = new LoadImgforString(msg.getSubject(),
                    path, true);
            ReadInFile.method2(pathDb, lifs.getAimString());
            //向 Messages 类中的一些属性赋值
            msg.setName(StringHtml.converthtml(msg.getName()));
            msg.setHeadMsg(StringHtml.converthtml(msg.getHeadMsg()));
            msg.setSubject(pathDb.substring(pathDb.length() - 35));
            //这里有很多栏目的信息采用了这个方法执行添加操作,因此需要设置 Messages 类中
的个别属性值。
            //如果不是新闻中心信息，无一级菜单
            if(msg.getMsgType()<7){
                msg.setParentId(0);
            }else {
                msg.setParentId(1);
            }
            //获取发表新闻的作者姓名
            String createUserString="";
            if(loginUser.getRealname()==""||loginUser.getRealname()==null){
```

```
                createUserString="佚名";
            }else{
                createUserString=loginUser.getRealname();
            }
            msg.setCreateUser(createUserString);
            //是否置顶，默认不置顶
            msg.setSign(0);
            //创建时间为当前时间
            msg.setCreateTime(new Date());
            //调用业务层方法，执行添加操作
            msgService.addMessage(msg);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return "addMsg";
    }
```

测试程序，单击新闻中心管理首页界面上的“添加新闻”按钮，打开如图 14-9 所示的界面。输入信息，单击“保存信息”按钮，添加新闻成功。

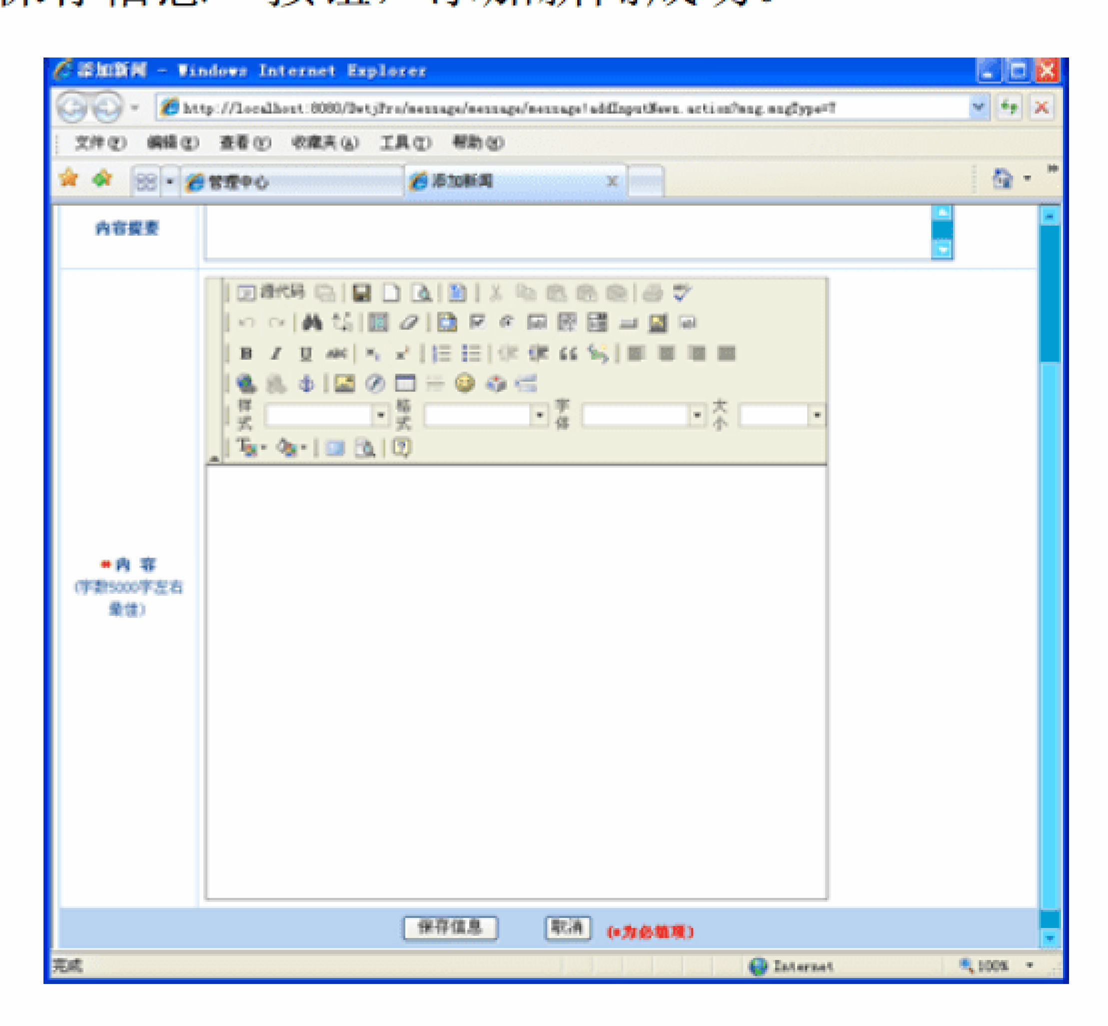


图 14-9　添加新闻界面

14.3.3　修改新闻信息

修改和删除功能的实现非常简单，下面以修改为例，讲解一下其实现思路。

(1) 在新闻中心管理首页中的“编辑”按钮中添加超链接：message/message!updateInput.action?msg.id=<s:property value="#msg.id"/>。读者需要把新闻 Id 传过去作为标识，在 MessageAction 类中编辑 updateInput()方法，代码如下。

```
    //打开修改信息界面
    public String updateInput() {
        SessionFilterAction.getUser();
        //获取 HttpServletRequest 对象
```

```
        HttpServletRequest request = ServletActionContext.getRequest();
        //根据新闻 Id 获取特定的新闻数据
        Messages msgs = msgService.getMessageById(msg.getId());// 获取特定的信息
        //读取新闻内容信息
        String pathDb = ServletActionContext.getServletContext().getRealPath(
                msgs.getSubject());
        msgs.setSubject(ReadOutFile.readFile(pathDb));
        //把要修改的信息保存至 HttpServletRequest 对象中
        request.setAttribute("msg", msgs);
        return "updateMsg";
    }
```

(2) 在 struts-config 文件夹下的 message.xml 文件中配置“updateMsg”的结果页面，配置代码如下。

```
<result name="updateMsg">/admin/message/updateInput.jsp</result>
```

(3) 编辑 updateInput.jsp 页面，加载特定的新闻信息，代码如下。

```
<form action="message/message!updateMessage.action" method="post"
enctype="multipart/form-data" name="myform">
    <table width="90%" border="0" align="center" cellpadding="5"
cellspacing="1" class="table">
        <input type="hidden" name="msg.id" value="<s:property
value="#request.msg.id"/>" >
        <input type="hidden" name="msg.langType" value="<s:property
value="#request.msg.langType"/>"/>
        <input type="hidden" name="msg.msgType" value="<s:property
value="#request.msg.msgType"/>"/>
        <input type="hidden" name="msg.clickNum" value="<s:property
value="#request.msg.clickNum"/>"/>
        <input type="hidden" name="msg.tempStr" value="<s:property
value="#request.msg.tempStr"/>"/>
        <input type="hidden" name="msg.sign" value="<s:property
value="#request.msg.sign"/>"/>
        <tr>
            <th colspan="2" height="30px" class="STYLE5" align="left">更新信
息</th>
        </tr>
        <tr>
            <td height="25" align="center" class="td_bg">
                <strong><font color="#FF0000">*</font>信息标题</strong>
            </td>
            <td width="86%" class="td_bg">
                <input type="text" name="msg.name" size="40" id="titname"
maxlength="30" value="<s:property value="#request.msg.name"/>"
onblur="checktitle()"/>
            </td>
        </tr>
        <tr>
```

```
            <td height="25" align="center" class="td_bg">
                <strong>内容提要</strong>
            </td>
            <td height="25" align="left" class="td_bg">
                <textarea rows="2" cols="83" name="msg.headMsg" id="headmsg"
onblur="checkhead()"><s:property value="#request.msg.headMsg"/></textarea>
            </td>
        </tr>
            <tr>
                <td height="25" align="center" class="td_bg">
                    <strong><font color="#FF0000">*</font>内 容<br>
</strong>(字数 5000 字左右最佳)
                </td>
                <td class="td_bg">
                        <s:property value="#request.msg.subject"
escape="false"/>
                </td>
            </tr>
            <tr>
                <th align="center" colspan="2">
                    <input type="submit" value="更新信息">  
                    <input type="button" value="取消"
onclick="javascript:window.close();">
                </th>
            </tr>
        </table>
</form>
```

(4) 配置以后，当单击新闻中心首页中的“编辑”按钮时，页面跳转至“admin/message/updateInput.jsp”页面，在这个页面中以表单的形式输出新闻信息，以供管理员修改。修改后，单击页面中的“更新信息”按钮，提交修改表单至 MessageAction 类中的 updateMessage() 方法，该方法可以实现修改新闻信息功能，代码如下。

```
    //更新消息
    public String updateMessage() {
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        try {
            //重新把修改后的新闻内容生成页面
            String pathDb = ServletActionContext.getServletContext()

    .getRealPath(RandomStringtable.genRandomNum("upLoadDB/"));
            ReadInFile.method2(pathDb, msg.getSubject());
            msg.setSubject(pathDb.substring(pathDb.length() - 35));
            //获取登录用户信息
            Users users = (Users) request.getSession().getAttribute("login");
            //设置修改时间
            msg.setCreateTime(new Date());
            //设置修改者
```

```
            String sourcenameString="";
            if(users.getRealname()==""||users.getRealname()==null){
                sourcenameString="佚名";
            }else {
                sourcenameString=users.getRealname();
            }
            msg.setCreateUser(sourcenameString);
            //设置新闻名称
            msg.setName(StringHtml.converthtml(msg.getName()));
            //设置新闻导读
            msg.setHeadMsg(StringHtml.converthtml(msg.getHeadMsg()));
                    if(msg.getMsgType()<7){
                msg.setParentId(0);
            }else {
                msg.setParentId(1);
            }
            //调用业务层方法，执行修改操作
            msgService.updateMessage(msg);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return "updateSuccess";
    }
```

测试程序。单击新闻中心管理首页界面上的“编辑”按钮，查询特定的信息，并加载至 message 文件夹下的 updateInput.jsp 页面上，运行效果如图 14-10 所示。



图 14-10　新闻更新界面

新闻的删除和修改大同小异，思路完全一样，这里不再讲述。

新闻中心的二级栏目中还有一个疑惑解答栏目，它的新闻信息存放在独立的表中(Questions 表)，功能的实现和上面所讲到的思路完全一样，这里也不再讲述，读者可以查看案例源码进行进一步的了解。

14.4 前台展示——新闻中心

新闻中心栏目分为太极动态、理论天地、太极养生、武林资讯和疑惑解答 5 个栏目，也就是说这 5 个栏目需要在前台展现出来。为了简单，这里的栏目是固定的，也就是说不需要从数据库中读取栏目，只要在页面上显示即可。

14.4.1 获取二级栏目的新闻信息

在前台浏览新闻中心下面的信息时，需要把新闻中心的二级栏目下的所有新闻查询并展示出来。

(1) 在 MessageAction 类中编辑 getSecondMenu()方法，调用业务层的方法分别查询出太级动态、理论天地、太极养生、武林资讯和疑惑解答栏目下的新闻信息，方法的实现代码如下。

```
    //点击新闻中心，获取二级菜单信息
    public String getSecondMenu(){
        HttpServletRequest request=ServletActionContext.getRequest();
        //调用业务层方法查询太极动态栏目
        List<Messages> dongList=msgService.getMessage(5, msg.getLangType(),
7);
        request.setAttribute("dongs", dongList);
        //调用业务层方法查询理论天地
        List<Messages> li=msgService.getMessage(5, msg.getLangType(), 8);
        request.setAttribute("li", li);
        //调用业务层方法查询太极养生
        List<Messages> yangList=msgService.getMessage(5, msg.getLangType(),
9);
        request.setAttribute("yanglist", yangList);
        //调用业务层方法查询武林资讯
        List<Messages> wuList=msgService.getMessage(5, msg.getLangType(),
10);
        request.setAttribute("wulist", wuList);
        //查询疑惑解答问题
        List<Questions> quList=questionService.getQuestionList();
        request.setAttribute("questions", quList);
        request.getSession().setAttribute("lang",msg.getLangType());//获取语言
        return "menu";
    }
```

上面代码调用了业务层 MessageServiceImpl 类中的 getMessage(int num,String langType, int msgType)方法，获取了太级动态、理论天地、太极养生和武林资讯 4 个栏目的新闻信息。getMessage(int num,String langType,int msgType)方法有三个参数，其中，第一个参数表示要显示的数据条数；第二个参数表示显示的信息是中文还是英文；第三个参数表示要显示的新闻是哪个栏目的。这样把三个参数传递给业务层方法后，业务层方法会根据传过去的语言类型(第

二个参数)、信息类型(第三个参数)查询出符合条件的前 5 条数据，并返回一个 List<Messages>集合。疑惑解答的信息获取无需参数，只是查询出来的结果数据也是根据语言类型，并且也是只查询了前 5 条数据。

(2) 在 struts-config 文件夹下的 message.xml 文件中配置“menu”的结果页面，配置如下。

```
<result name="menu">/news_menu.jsp</result>
```

(3) 在根目录下新建 news_menu.jsp 页面，把查询出来的结果数据显示出来。这里只把太极动态栏目下的新闻信息显示代码贴出来。

```
<s:if test="#session.lang.equals(\"zh\")">
        太极动态
</s:if>
<s:else>
        Tai Ji Dynamic
</s:else>
<ul class="ul">
    <s:if test="#request.dongs.size()>0">
        <s:iterator value="#request.dongs" var="dong">
            <li>
                <img src="images/new.gif" width="30" height="8"/>
                <a style="clear:both;cursor: pointer;"
href="message/message!findNewById.action?msg.id=<s:property
value="#dong.id"/>&msg.langType=<s:property
value="#session.lang"/>&sign=news">
                    <s:if test="#dong.name.trim().length()>8">
                        <s:property value="%{#dong.name.substring(0,8)}" />...
                    </s:if>
                    <s:else>
                        <s:property value="#dong.name" />
                    </s:else>
                </a>
            </li>
        </s:iterator>
    </s:if>
    <s:else>
        <li>
            <s:if test="#session.lang.equals(\"zh\")">暂无数据</s:if>
            <s:else>Sorry! This category have nothing data.</s:else>
        </li>
    </s:else>
</ul>
```

理论天地、太极养生、武林资讯和疑惑解答下的新闻展示代码和上面代码大同小异，具体代码的实现请查看本案例源码。

(4) 在网站中文首页导航中的“新闻中心”栏目中添加超链接：message/message!getSecondMenu.action?msg.langType=zh，英文首页导航中添加超链接：message/message!getSecondMenu.action?msg.langType=en。

运行程序，新闻中心列表页面如图 14-11 所示。

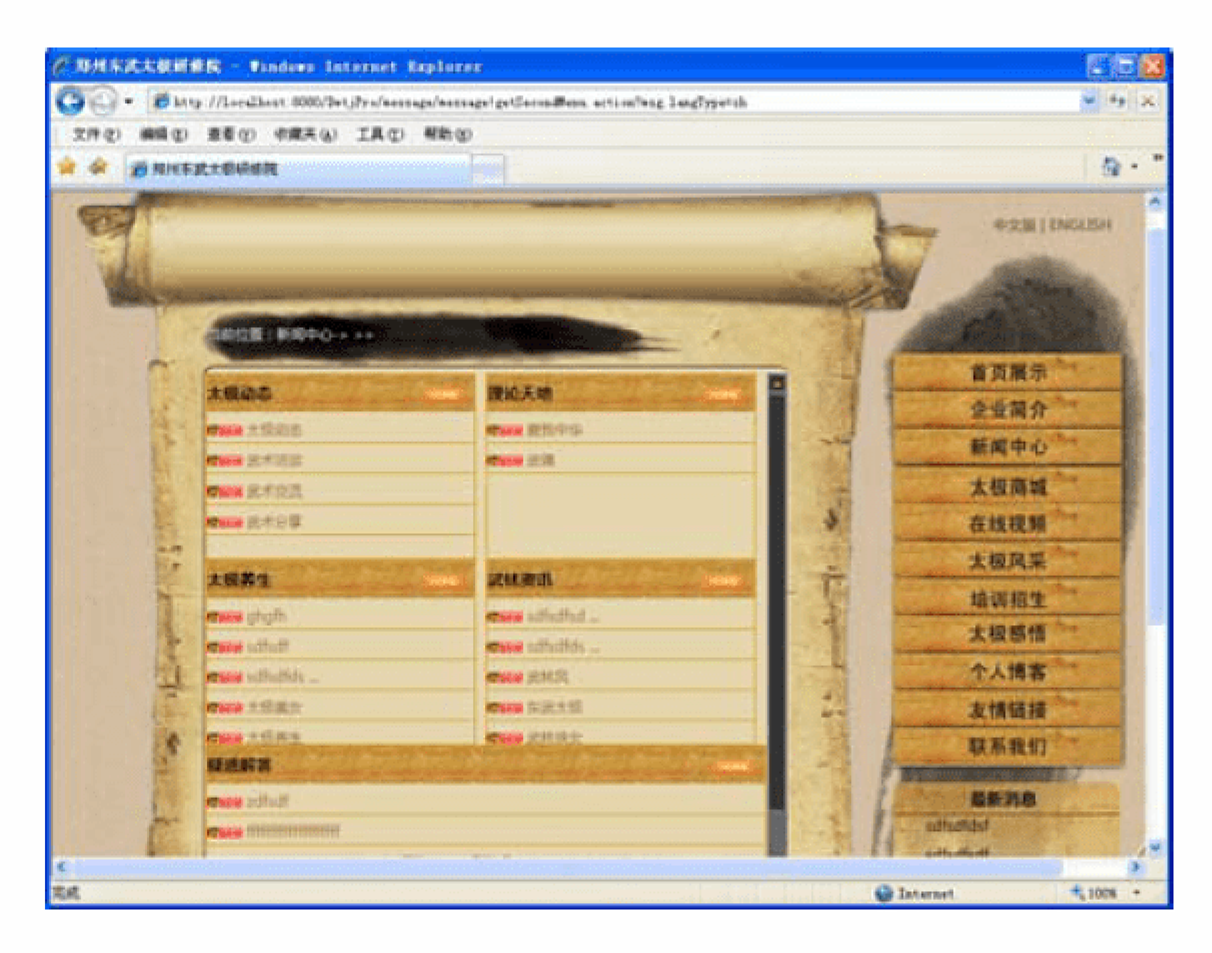


图 14-11　获取二级栏目的新闻信息界面

14.4.2　获取特定的新闻信息

读者在上面代码中看到，在显示新闻列表时，对新闻标题(名称)设置了一个超链接：message/message!findNewById.action?msg.id=<s:property value="#dong.id"/>&msg.langType=< s:property value="#session.lang"/>&sign=news，也就是说，当浏览者单击新闻列表中的新闻标题时，系统访问 MessageAction 类中的 findNewById()方法，并把新闻 Id 作为参数传过去。

(1)　在 MessageAction 类中编辑 findNewById()方法，代码如下。

```
    //获取具体的新闻信息
    public String findNewById() {
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        Messages news = msgService.getMessageById(msg.getId());
                                                    // 获取特定的新闻信息
        request.getSession().setAttribute("lang", msg.getLangType());
                                                    // 获取语言
        //把查询到的特定新闻保存至 HttpServletRequest 对象中
        request.setAttribute("news", news);
        request.getSession().setAttribute("sign",
request.getParameter("sign"));
        //当阅读特定新闻时，点击率加 1
        news.setClickNum(msgService.getCountClick(msg.getId())+1);
        //调用业务层方法，更新点击率
        msgService.updateMessage(news);
        return "news";
    }
```

(2)　在 struts-config 文件夹下的 message.xml 文件中配置“news”的结果页面，配置代码如下。

```
<result name="news">/news_show.jsp</result>
```

(3) 在系统根目录下新建 news_show.jsp 页面，显示特定的新闻信息，代码片段如下。

```
信息来源：
            <s:if test="#request.news.tempStr.equals(\"zi\")">本站原创</s:if>
            <s:else>转载信息</s:else>
更新时间：<s:date name="#request.news.createTime" format="yyyy-MM-dd"/> 作者：
<s:property value="#request.news.createUser"/> 访问量： <s:property
value="#request.news.clickNum"/>
新闻内容：<jsp:include
page='<%=((Messages)request.getAttribute("news")).getSubject() %>'/>
```

单击新闻列表页面中的新闻列表中的新闻标题，页面跳转至 news_ show.jsp 页面，如图 14-12 所示。



图 14-12　前台查阅特定的新闻信息

14.5　后台模块——太极商城

太极商城栏目下又存在音像制品、太极服装、太极器械和太极书籍 4 个二级栏目，在商品数据表(Products)中存在一个“msgType”字段，此字段标识商品信息栏目：1 表示音像制品、2 表示太极服装、3 表示太极器械、4 表示太极书籍。

14.5.1　查询商品信息，分页显示

(1) 在 com.dwtj.actions 包下新建 ProductAction 类，继承自 com.opensymphony.xwork2. ActionSupport 类，并重写父类的 execute()方法，在该方法中调用商品业务层类中的方法，根据不同的商品栏目查询该栏目下的所有信息，分页显示。execute()方法代码如下。

```
    //查询所有的商品信息，分页显示
    public String execute() throws Exception {
        //获取 HttpServletRequest 对象
```

```
        HttpServletRequest request = ServletActionContext.getRequest();
        PageBean pageBean = (PageBean) request.getSession().getAttribute(
                "pageBean");
        if (pageBean == null) {
            pageBean = new PageBean();
        }
        String mark = "";
        if (request.getParameter("mark") != null) {
            mark = request.getParameter("mark");
        } else {
            mark = "f";
        }
        int tempNumber = 0;
        if (request.getParameter("number") != null) {
            tempNumber = Integer
                    .parseInt(request.getParameter("number").trim());
        }
        // 调用业务层分页显示数据
        try {
            pageBean =
productService.selectProductsAll(PageCtr.PageCtrUpdate(
                    pageBean, mark, tempNumber), 20, "",product.getMsgType());
        } catch (Exception e) {
            e.printStackTrace();
        }
        //把页面中传过来的商品栏目标示储存起来
        request.setAttribute("msgType", product.getMsgType());
        //把获取到的符合条件的商品信息存放在 HttpServletRequest 对象中
        request.setAttribute("products", pageBean.getResult());
        pageBean.setResult(null);
        request.getSession().setAttribute("pageBean", pageBean);
        return "productIndex";
    }
```

(2) 在 struts-config 文件夹下新建 product.xml 文件，配置 ProductAction 类，并配置“productIndex”的结果页面，配置代码如下。

```
<package name="product" namespace="/product" extends="ma">
        <action name="product" class="com.dwtj.actions.ProductAction" >
            <result name="productIndex">/admin/product/index.jsp</result>
        </action>
</package>
```

(3) 在后台管理系统中的左边导航菜单中，设置太极商城模块的二级栏目超链接，路径如下所示。

```
<a  href="product/product.action?product.msgType=1"  target="main">音像制品
</a>
<a  href="product/product.action?product.msgType=2"  target="main">太极服装
</a>
```

```
<a  href="product/product.action?product.msgType=3"  target="main">太极器械
</a>
<a  href="product/product.action?product.msgType=4"  target="main">太极书籍
</a>
```

(4)　在 admin 文件夹下新建 product 文件夹，并在 product 文件夹下新建 index.jsp 页面，把商品信息展示出来，代码如下。

```
<table class="table" cellspacing="1" cellpadding="2" width="100%"
align="center" border="0">
    <tr>
        <th height="25" colspan="4" align="left" class="man">
            <s:if test="#request.msgType==1">音像制品</s:if>
            <s:elseif test="#request.msgType==2">太极服装</s:elseif>
            <s:elseif test="#request.msgType==3">太极器械</s:elseif>
            <s:else>太极书籍</s:else>
        </th>
        <th height="25" align="right" class="addMsg" colspan="2">添加商品</th>
    </tr>
    <tr>
        <td class="td bg" width="15%" height="25">商品名称</td>
        <td width="15%" class="td bg">商品价格</td>
        <td class="td bg" width="15%">商品编码</td>
        <td class="td_bg" width="15%">语言</td>
        <td class="td_bg" width="15%">编辑</td>
        <td class="td_bg" width="15%">删除</td>
    </tr>
    <s:if test="#request.products.size()!=0">
        <s:iterator value="#request.products" var="product">
            <tr>
                <td class="td_bg" width="15%" height="25">
                    <s:if test="#product.name.trim().length()>20">
                        <s:property value="%{#product.name.substring(0,20)}"
/>…
                    </s:if>
                    <s:else>
                        <s:property value="#product.name"/>
                    </s:else>
                </td>
                <td class="td_bg" width="15%">
                    <s:property value="#product.price"/>
                    </td>
                    <td class="td_bg" width="15%">
                        <s:property value="#product.number"/>
                    </td>
                    <td class="td_bg" width="15%">
                        <s:if test="#product.langType.equals(\"zh\")">中文
</s:if>
                        <s:else>英文</s:else>
                    </td>
                    <td class="td_bg" width="15%">【编辑】</td>
```

```
                    <td class="td_bg" width="15%">【删除】</td>
                </tr>
            </s:iterator>
        </s:if>
        <s:else>
            <tr>
                <td colspan="6" class="td bg" height="25px">暂无商品信息</td>
            </tr>
        </s:else>
        <s:if test="#request.products.size!=0">
            <tr>
                <th class="bg_tr" align="right" colspan="6" height="25">
                    <s:push value="#request.session.pageBean">
                    共 <span class="red"><s:property value="totalCount"
/></span>条记录  当前第<span class="red"><s:property value="nowpage+1"
/></span>/<s:property value="totalpage" />页 每页<s:property
value="pagesize" />条数据
                    </s:push>
                    <s:a
href="product/product.action?product.msgType=%{#request.msgType}&mark=f&num
ber=0">首页</s:a>
                    <s:a
href="product/product.action?product.msgType=%{#request.msgType}&mark=p&num
ber=0">上一页</s:a>
                    <s:a
href="product/product.action?product.msgType=%{#request.msgType}&mark=n&num
ber=0">下一页</s:a>
                    <s:a
href="product/product.action?product.msgType=%{#request.msgType}&mark=l&num
ber=0">尾页</s:a>
                </th>
            </tr>
    </s:if>
</table>
```

单击左边导航太极商城模块下的“音像制品”，出现如图 14-13 的页面效果。

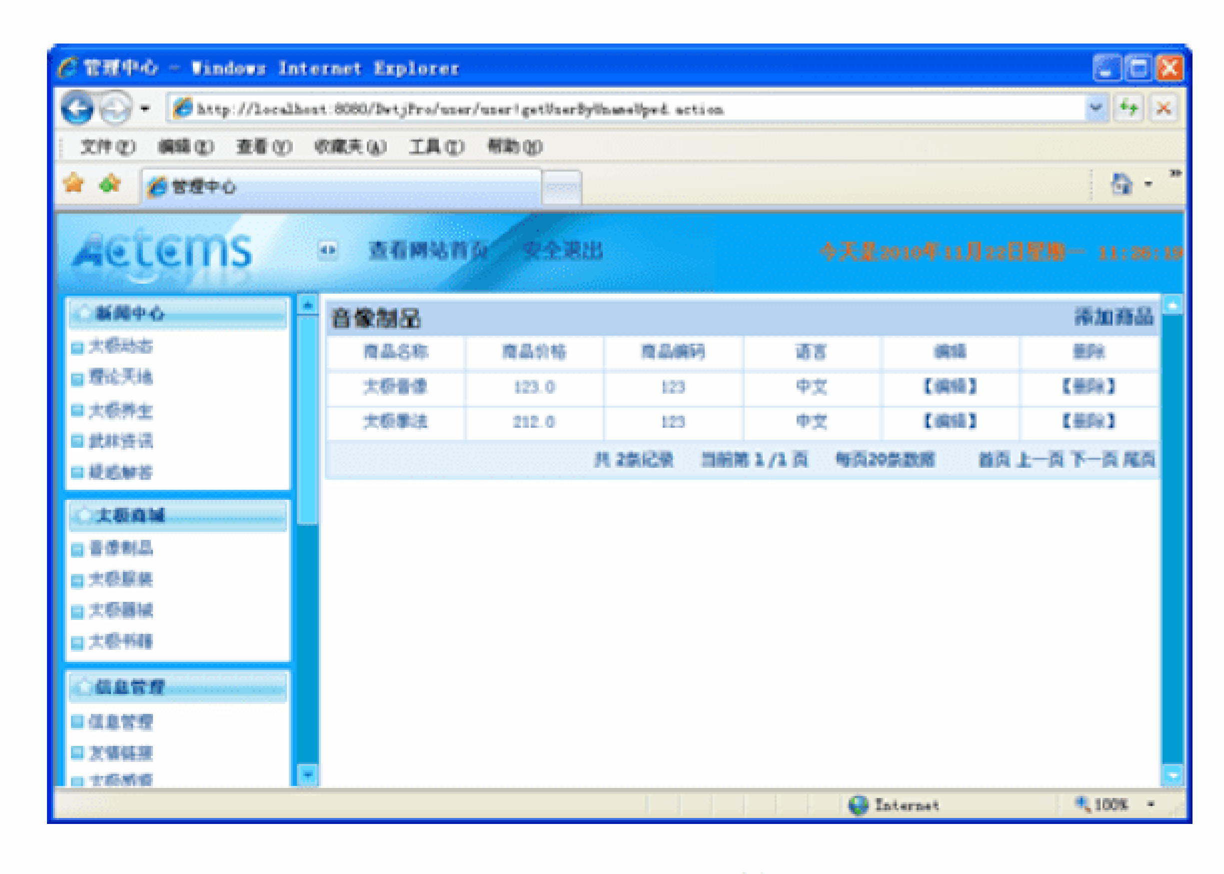


图 14-13　太极商城管理首页

14.5.2　添加商品信息

从上图 14-13 中可以看到，在太极商城管理首页右上角有一个“添加商品”按钮，这个按钮和新闻中心模块中的新闻添加的实现一样。

(1) 在太极商城管理首页中的“添加商品”按钮上添加“onclick”事件，代码如下。

```
<a
onclick="openWin('product/product!addInputProduct.action?product.msgType=<s
:property value="#request.msgType"/>','addInput',800,900,1);">添加商品</a>
```

(2) 通过上面的连接地址，单击“添加商品”按钮时，执行 ProductAction 类中的 addInputProduct()方法，这个方法的功能就是打开添加商品页面，方法的实现代码如下。

```
    //打开添加商品信息
    public String addInputProduct() {
        //获取 HttpServletRequest 对象
        HttpServletRequest request=ServletActionContext.getRequest();
        //把商品栏目保存至 HttpServletRequest 对象中
        request.setAttribute("msgType", product.getMsgType());
        return "addInputProduct";
    }
```

(3) 在 struts-config 文件夹下的 product.xml 文件中配置打开添加商品的结果页面，即“addInputProduct”的结果页面，配置如下。

```
<result name="addInputProduct">/admin/product/add_input.jsp</result>
```

(4) 单击太极商城管理首页右上角的“添加商品”按钮，打开如图 14-14 的页面。

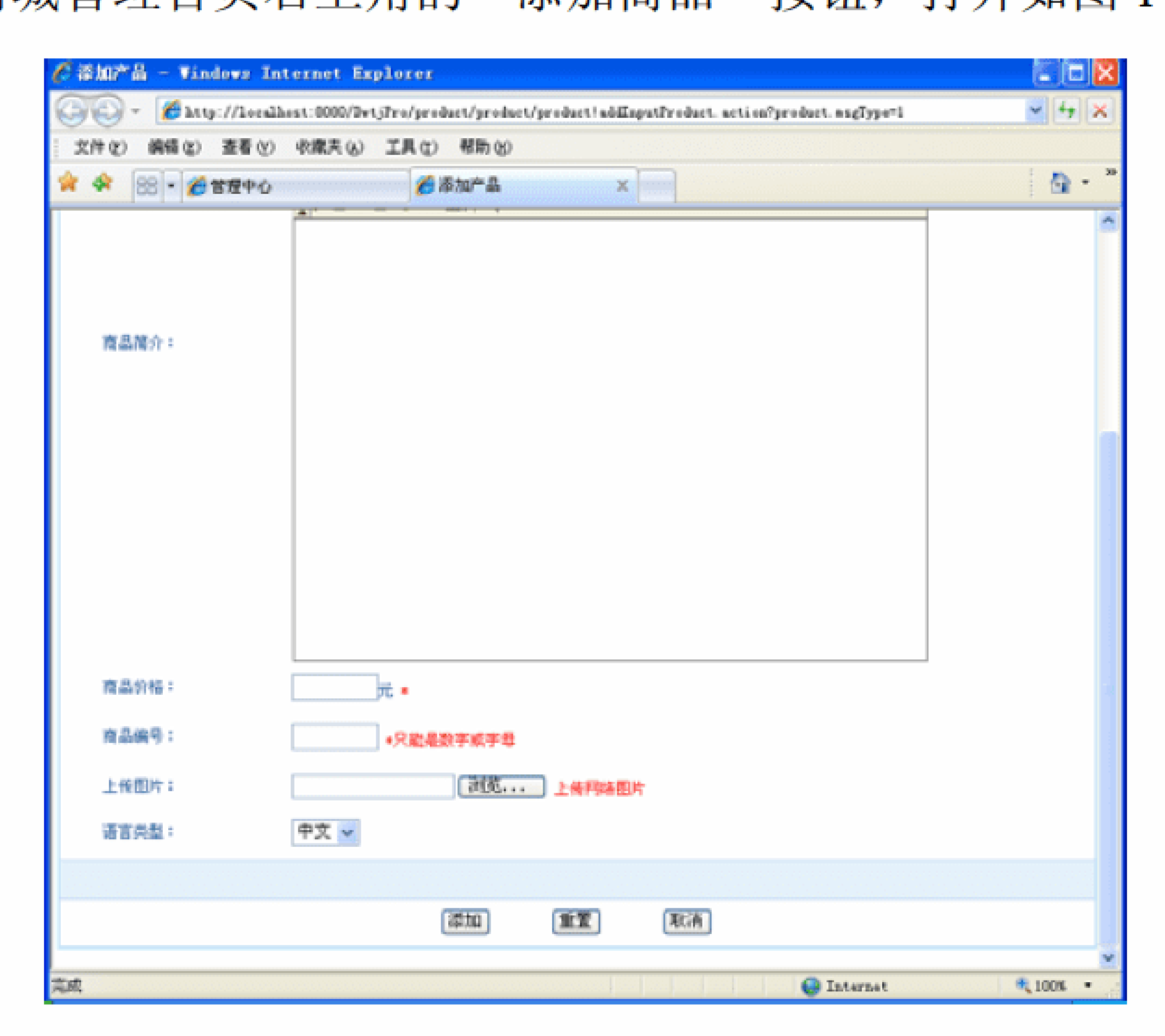


图 14-14　添加商品信息界面

(5) 在 ProductAction 类中添加 addProduct()方法，实现商品信息的添加功能。代码如下。

```
    //添加商品信息
    public String addProduct() {
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        try {
            // 获取文件路径，实现图片的上传功能
            String path =
ServletActionContext.getServletContext().getRealPath(
                    savePath);
            File imageFile = new File(path + "\\" + fileFileName);
            if(imageFile.toString().indexOf("null")==-1){
                System.out.println(imageFile);
                //定义一个上传文件的输入流
                BufferedInputStream bis = null;
                //定义一个上传文件的输出流
                BufferedOutputStream bos = null;
                try {
                    // 获取一个上传文件的输入流
                    bis = new BufferedInputStream(new FileInputStream(file));
                    // 获取一个上传文件的输出流
                    bos = new BufferedOutputStream(new
FileOutputStream(imageFile));
                    byte buf[] = new byte[(int) file.length()];
                    int length = 0;
                    while ((length = bis.read(buf)) != -1) {
                        bos.write(buf, 0, length);
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                } finally {
                    try {
                        if (bis != null) {
                            bis.close();
                        }
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    try {
                        if (bos != null) {
                            bos.close();
                        }
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
                //获取图片路径，并赋值
                product.setPic("proImages/" + imageFile.getName());
            }
```

```
            //把 Fckeditor 中的内容生成一个页面并保存至特定的路径下
            String path1=
ServletActionContext.getServletContext().getRealPath(
                    uploadDir);
            String pathDb = ServletActionContext.getServletContext()

    .getRealPath(RandomStringtable.genRandomNum("upLoadDB/"));
            LoadImgforString lifs = new LoadImgforString(product.getIntro(),
                    path1, true);
            ReadInFile.method2(pathDb, lifs.getAimString());
            product.setIntro(pathDb.substring(pathDb.length() - 35));
            //调用业务层方法，执行添加商品信息操作
            productService.addProduct(product);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return "addSuccess";
    }
```

到此，商品信息的添加功能就完成了，单击添加商品信息界面上的“添加”按钮时，执行 ProductAction 类中的 addProduct()方法，完成添加功能的操作。

14.5.3　删除商品信息

商品信息的更新和删除思路大同小异，前面在讲解后台管理系统中的新闻中心模块时已经提到过，都需要把特定信息的 Id 作为参数，传递给 Action 类中执行信息的更新或删除的实现方法中。

(1)　在 ProductAction 类中编辑 delProduct()方法，实现商品的删除操作，方法的实现代码如下。

```
    //删除商品信息
    public String delProduct() {
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        //调用业务层方法，实现商品信息的删除操作
        productService.deleteProduct(product.getId());
        SystemLogsCommon.getSystemLogsMessage(request, "删除商品信息");
        return "delSuccess";
    }
```

(2)　在 struts-config 文件夹下的 product.xml 文件中配置“delSuccess”的结果页面。当执行删除操作后可以跳转至 Action 类中的查询所有商品信息方法，即 execute()方法，读者可以根据需要设置结果页面。

(3)　在太极商城管理首页中的“删除”按钮中添加链接，代码如下所示。

```
<a style="cursor: pointer;"
onclick="del('product/product!delProduct.action?product.id=<s:property
value="#product.id"/>')">“删除”</a>
```

单击“删除”按钮，提示“你确认要删除记录吗？”信息，如图 14-15 所示。

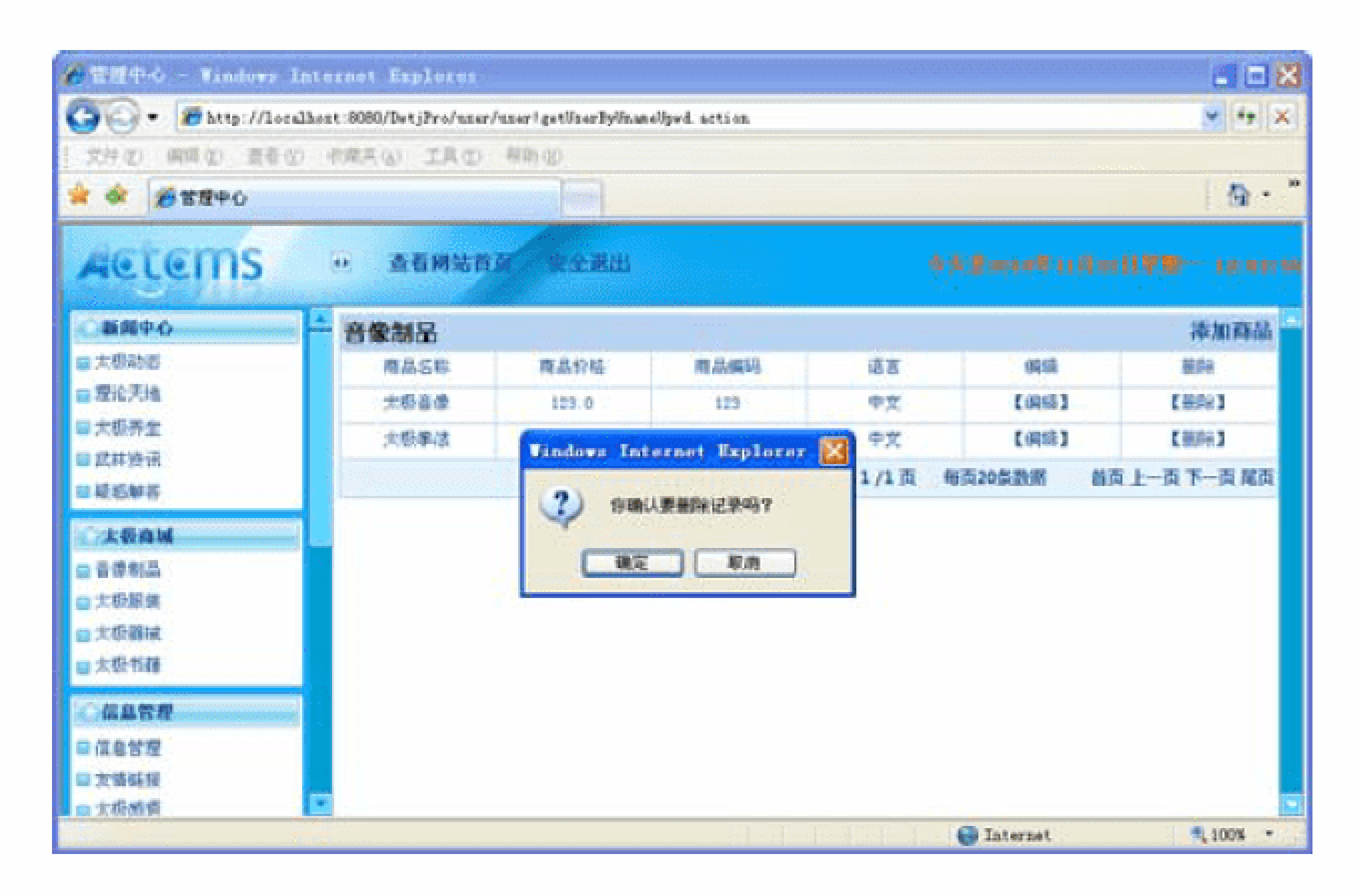


图 14-15　删除商品时的提示信息

14.6　前台展示——太极商城

太极商城栏目下存在音像制品、太极服装、太极器械和太极书籍 4 个二级栏目，这里和新闻中心的二级栏目一样，也是固定的，只需要把二级栏目下的商品信息读取到页面上展示出来即可。

14.6.1　获取二级栏目的商品信息

当单击前台首页中的“太极商城”栏目时，动态的读取二级栏目下的商品信息，并以列表的形式展示在页面中。

(1) 在 ProductAction 类中编辑 getMenu()方法，调用业务层方法 getproList(String langType,int msgType)获取商品信息列表，其中，“langType”参数表示语言类型、“msgType”表示栏目标识，即在业务层查询商品信息集合是根据语言类型和栏目来进行的，只查询出符合条件的前 7 条数据，并在页面上显示出来。getMenu()方法的实现代码如下。

```
    //获取二级菜单信息
    public String getMenu(){
        HttpServletRequest request=ServletActionContext.getRequest();
        request.getSession().setAttribute("lang", product.getLangType());
        //查询音像制品栏目下的商品信息
        List<Products>
yinsList=productService.getproList(product.getLangType(), 1);
        request.setAttribute("yins", yinsList);
```

```
        //查询太极服装栏目下的商品信息
        List<Products>
fuList=productService.getproList(product.getLangType(), 2);
        request.setAttribute("fus", fuList);
        //查询太极器械栏目下的商品信息
        List<Products>
qiList=productService.getproList(product.getLangType(), 3);
        request.setAttribute("qis", qiList);
        //查询太极书籍栏目下的商品信息
        List<Products>
shuList=productService.getproList(product.getLangType(), 4);
        request.setAttribute("shus", shuList);
        return "menupro";
    }
```

(2) 在 struts-config 文件夹下的 product.xml 文件中配置“menupro”的结果页面，配置如下。

```
<result name="menupro">/product_menu.jsp</result>
```

(3) 在项目的根目录下新建 product_menu.jsp 页面，把查询到的信息展现在页面上。代码如下。

```
<div id="hotNews" style="width: 230px; height: 200px; border-collapse: collapse;
border-width: 1px; border-color: #C90; border-style: solid; float: left;">
    <h3>
        <s:if test="#session.lang.equals(\"zh\")">音像制品</s:if>
        <s:else>Audio&VideoProducts</s:else>
    </h3>
    <ul class="ul">
        <s:if test="#request.yins.size()>0">
            <s:iterator value="#request.yins" var="yin">
                <li>
                    <s:if test="#yin.name.trim().length()>15">
                        <s:property value="%{#yin.name.substring(0,15)}" />…
                    </s:if>
                    <s:else><s:property value="#yin.name" /></s:else>
                </li>
            </s:iterator>
        </s:if>
        <s:else>
            <li>
                <s:if test="#session.lang.equals(\"zh\")">暂无数据</s:if>
                <s:else>Sorry! This category have nothing data.</s:else>
            </li>
        </s:else>
    </ul>
</div>
```

上面是显示音像制品栏目下的商品信息列表代码，显示太极服装、太极器械和太极书籍栏目下的商品信息实现代码和上面代码大同小异。商品列表页面如图 14-16 所示。

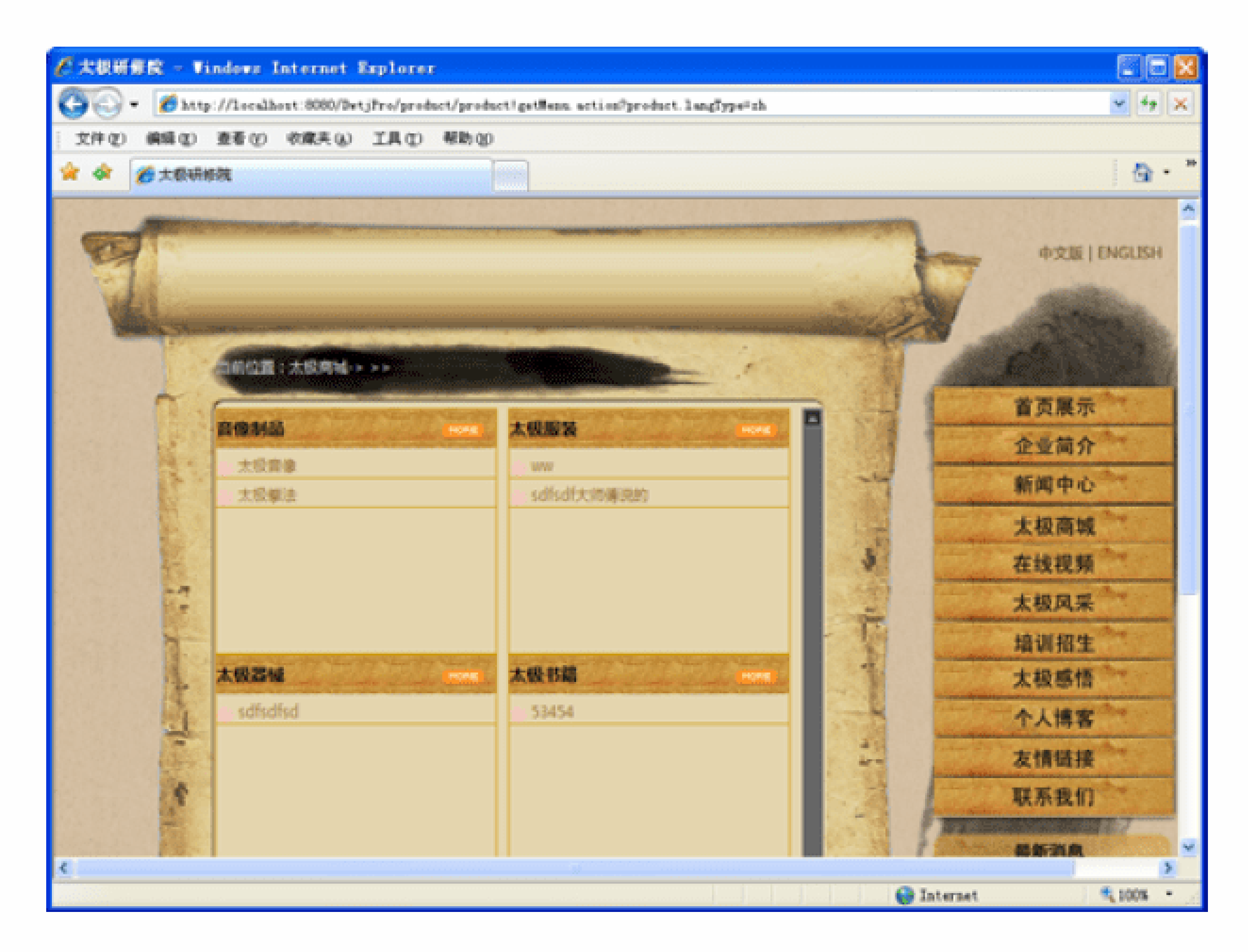


图 14-16　二级栏目下的商品信息展示页面

14.6.2　获取特定的商品信息

上面是以列表的信息展示了每个二级栏目下的商品信息，但是如果要查看具体的一条商品信息怎么办呢？以下是这一实现方式的操作步骤。

(1) 对商品名称设置超链接，例如对音像制品栏目下的商品名称设置超链接为：product/product!findProById.action?product.id=<s:property value="#shu.id"/>&product.langType=<s:property value="#session.lang"/>。其中，需要把商品信息的 Id 传给 ProductAction 类中的 findProById()方法，product.id 的值为<s:property value="#shu.id"/>，“shu”是页面中遍历“shus”集合时设置的 var 属性值。

(2) 在 ProductAction 中编辑 findProById()方法，根据传过来的商品 Id 查询特定的商品信息并展示在页面上，代码如下。

```
//前台点击商品名称，显示具体的商品信息
public String findProById(){
    //获取 HttpServletRequest 对象
    HttpServletRequest request = ServletActionContext.getRequest();
    //调用业务层方法，获取特定的商品信息
    Products products = productService.getProById(product.getId());
    //获取商品简介，读取页面
    String pathDb = ServletActionContext.getServletContext().getRealPath(
            products.getIntro());
    products.setIntro(ReadOutFile.readFile(pathDb));
    //把查询到的商品信息保存至 HttpServletRequest 对象中
    request.setAttribute("pro", products);
    //保存语言类型
    request.getSession().setAttribute("lang",product.getLangType());
    return "thePro";
}
```

(3) 在 struts-config 文件夹下的 product.xml 文件中配置“thePro”的结果页面，配置如下。

```
<result name="thePro">/product.jsp</result>
```

(4) 在根目录下新建 product.jsp 页面，显示 HttpServletRequest 对象中保存的“pro”所对应的商品信息。代码片段如下。

```
<table cellpadding="0" cellspacing="1" border="1" class="table" width="430px"
height="420px">
    <s:if test="#request.pro!=null">
        <tr>
            <td width="150" height="250">
                <!----商品图片-->
                <div style="width: 150px; clear: both; height: 200px;">
                    <img src="<s:property value="#request.pro.pic" />"
width="150" height="150" />
                </div>
                <!----商品图片结束-->
            </td>
            <td>
                <ul>
                    <li class="liGoods">
                        <s:if test="#session.lang.equals(\"zh\")">商品名称:
</s:if>
                        <s:else>Product Name:</s:else>
                        <s:property value="#request.pro.name" />
                    </li>
                    <li class="liGoods">
                        <s:if test="#session.lang.equals(\"zh\")">商品编号:
</s:if>
                        <s:else>Product Number:</s:else>
                        <s:property value="#request.pro.number" />
                    </li>
                    <li class="liGoods">
                        <s:if test="#session.lang.equals(\"zh\")">商品价
格:</s:if>
                        <s:else>Product Price:</s:else>
                        <s:property value="#request.pro.price" />元
                    </li>
                    <li class="liGoods">
                        特色服务: <img src="images/service.gif" width="25"
height="21" style="display: inline; vertical-align: middle;" />
                    </li>
                    <li class="showTip">
                        <img src="images/service.gif" width="30" height="26"
style="display: inline; vertical-align: middle;" />
                        品质保证，无理由退换货!
                    </li>
                </ul>
            </td>
```

```
        </tr>
        <tr>
            <td colspan="2" style="border: 1px; border-color: #000; height:
180px; vertical-align: top;">
                <p style="word-break: break-all; vertical-align: top;">
                    <!-----简介-->
                        <s:property value="#request.pro.intro"
escape="false"/>
                    <!-----简介结束-->
                </p>
            </td>
        </tr>
    </s:if>
    <s:else>
        <tr>
            <td>
                <s:if test="#session.lang.equals(\"zh\")">暂无商品</s:if>
                <s:else>No products </s:else>
            </td>
        </tr>
    </s:else>
</table>
```

商品列表页面中的商品名称，跳转至 product.jsp 页面，如图 14-17 所示。

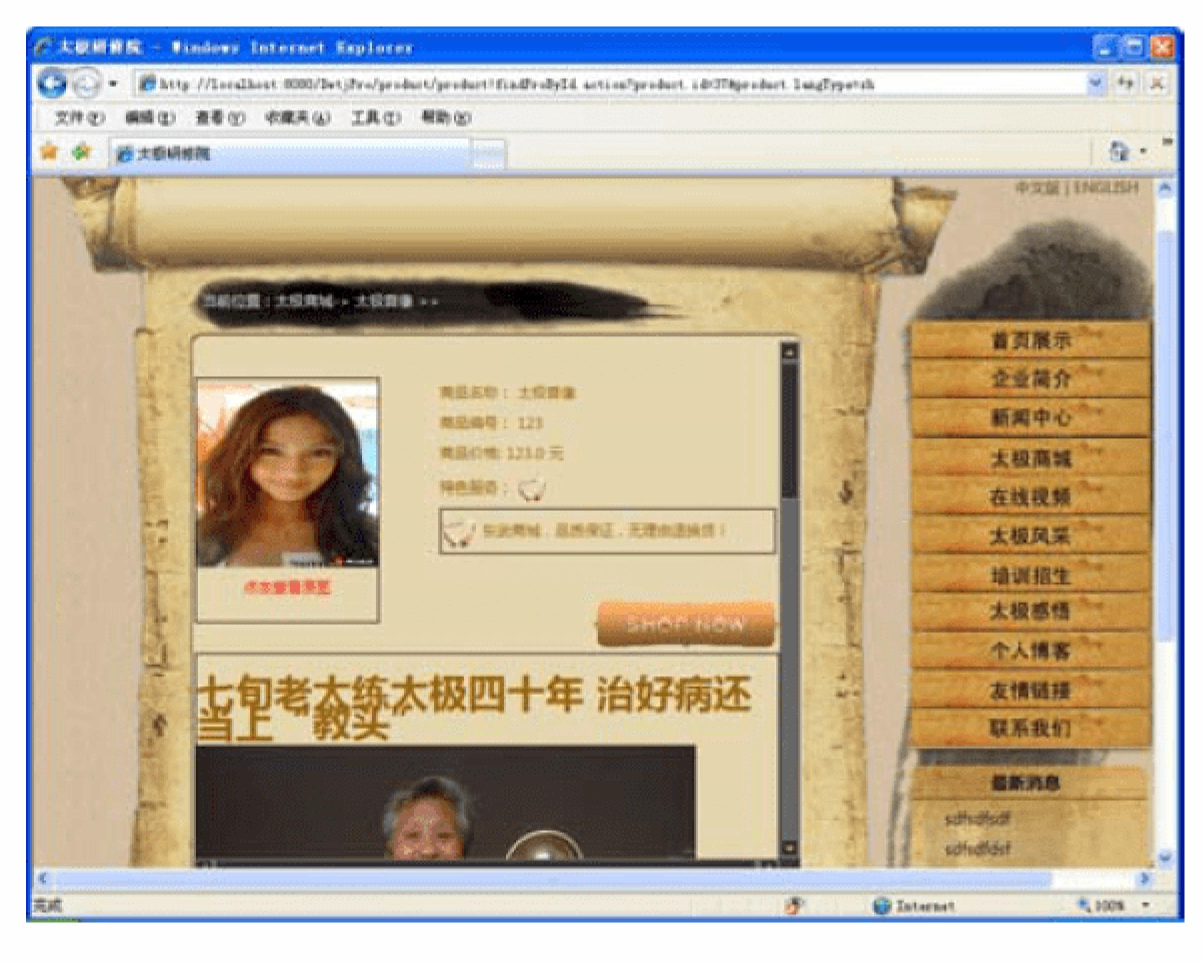


图 14-17　查看特定的商品信息

14.7　后台模块——信息管理

这个模块包含了企业简介、培训招生、在线视频、联系我们、友情链接、太极感悟和太极风采 7 个栏目的信息管理，根据数据表结构可以把将这 7 个栏目规划为信息管理、友情链接、太极感悟和太极风采 4 个模块进行管理。

14.7.1　信息管理

根据数据表结构可以将企业简介、培训招生、在线视频和联系我们 4 个栏目信息规划到信息管理模块中来进行统一管理。

前面提到 Messages 表中包含了多个栏目信息，其中包括企业简介、培训招生、在线视频和联系我们 4 个栏目，这 4 个栏目的信息结构是相同的，只需要用“msgType”字段来区分不同的栏目即可，其中，msgType 为 2 表示的是企业简介栏目、3 表示的是联系我们栏目、4 表示的是培训招生栏目、5 表示的是在线视频栏目。

(1)　在 MessageAction 类中重写 com.opensymphony.xwork2.ActionSupport 类中的 execute() 方法，在该方法中调用业务层方法，实现条件查询功能，代码如下。

```
    //查询所有的信息，包括企业简介、在线视频、联系我们和培训招生 4 个栏目信息
    public String execute() throws Exception {
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        PageBean pageBean = (PageBean) request.getSession().getAttribute(
                "pageBean");
        if (pageBean == null) {
            pageBean = new PageBean();
        }
        String mark = "";
        if (request.getParameter("mark") != null) {
            mark = request.getParameter("mark");
        } else {
            mark = "f";
        }
        int tempNumber = 0;
        if (request.getParameter("number") != null) {
            tempNumber = Integer
                    .parseInt(request.getParameter("number").trim());
        }
        String hqlString = "";// 存储不同的 sql 语句
        String signString="";//存储选中的一级菜单
        String sign="";//页面跳转时要传的参数
        //如果是根据不同的条件进行查询的信息

    if(request.getParameter("sign")!=null&&request.getParameter("sign")!=""
)
        {
            //如果根据栏目查询不同的信息
            if (request.getParameter("msgString") !=
null&&request.getParameter("msgString")!="") {
                request.setAttribute("msgString",
request.getParameter("msgString"));//保存选中的所属栏目
                //记录条件类型
```

```
                signString="msgType";
                //记录参数
                sign="msgString="+request.getParameter("msgString");
                hqlString = " where msgType="
                        +
Integer.parseInt(request.getParameter("msgString"));

            }
            //如果根据语言查询不同的信息
            else if (request.getParameter("langString") !=
null&&request.getParameter("langString")!="") {
                signString="langType";
                sign="langString="+request.getParameter("langString");
                request.setAttribute("langString",
request.getParameter("langString"));
                //如果用户选择的查询条件是“栏目”，则查询企业简介、培训招生、联系我们和
在线视频 4 个栏目中的信息
                hqlString = " where langType='"
                        + request.getParameter("langString") + "' and msgType
in (2,3,4,5)";

            }
            //调用业务层方法，获取查询到的信息，分页显示
            pageBean =
msgService.selectMessagesNewsAll(PageCtr.PageCtrUpdate(
                    pageBean, mark, tempNumber), hqlString);
        }
        //如果不根据条件查询，查询全部信息
        else {
            pageBean =
msgService.selectMessagesNewsAll(PageCtr.PageCtrUpdate(
                    pageBean, mark, tempNumber), "", "");
            signString="all";
        }
        //把页面跳转至不同页码时传过来的参数保存至 HttpServletRequest 对象中
        request.setAttribute("sendpage", request.getParameter("sign"));
        //把新赋值后的 sign 保存至 HttpServletRequest 对象中
        request.setAttribute("sign", sign);
        //把条件类型保存至 HttpServletRequest 对象中
        request.setAttribute("signString", signString);
        //把查询到的信息保存至 HttpServletRequest 对象中
        request.setAttribute("msgs", pageBean.getResult());
        pageBean.setResult(null);
        request.getSession().setAttribute("pageBean", pageBean);
        return "msgIndex";
    }
```

(2) 在 struts-config 文件夹下的 message.xml 文件中配置“msgIndex”的结果页面，配置如下。

```
<result name="msgIndex">/admin/message/index.jsp</result>
```

(3) 在 message 文件夹下新建 index.jsp 页面，显示查询到的结果数据列表。代码片段如下。

```
<table class="table" cellspacing="1" cellpadding="2" width="100%"
align="center" border="0">
    <tr>
        <th height="25" colspan="8" align="left" class="man" style="font-size:
14px;color: #1E5494;">查询条件:
            <select name="sign" id="sign" onchange="search()">
                <option value="0">---请选择查询条件---</option>
                <option value="msgType">所属栏目</option>
                <option value="langType">语    言</option>
                <option value="all">全  部</option>
            </select>
            <div id="divmsg" style="display: none;">
                <select name="msg" id="msg" onchange="checkmsg(this.value)">
                    <option value="0">---请选择所属栏目---</option>
                    <s:iterator value="{'企业简介','联系我们','培训招生','在线视
频'}" status="vs" var="item">
                        <option value="<s:property value="#vs.count+1"/>"
                            <s:if test="#request.msgString==#vs.count+1">
                                selected="selected"
                            </s:if>
                        ><s:property value="#item"/></option>
                    </s:iterator>
                </select>
            </div>
            <div id="divlang" style="display: none;">
                <select name="lang" id="lang"
onchange="checklang(this.value)">
                    <s:if test="#request.langString.equals(\"zh\")">
                        <option value="zh" selected="selected">---中文
---</option>
                        <option value="en">---英文---</option>
                    </s:if>
                    <s:elseif test="#request.langString.equals(\"en\")">
                        <option value="zh">---中文---</option>
                        <option value="en"  selected="selected">---英文
---</option>
                    </s:elseif>
                    <s:else>
                        <option value="0" selected="selected">---请选择语言类型
---</option>
                        <option value="zh">---中文---</option>
                        <option value="en">---英文---</option>
                    </s:else>
                </select>
            </div>
            <div id="divsign" style="display: inline;color: red;"></div>
        </th>
```

```
        </tr>
        <tr>
            <th height="25" colspan="4" align="left" class="man">
                信息管理
            </th>
            <th height="25" align="right" class="addMsg" colspan="4">
                添加信息</a>
            </th>
        </tr>
        <tr>
            <td class="td_bg" width="15%" height="25">
                <div align="center">信息标题</div>
            </td>
            <td class="td_bg" width="15%">
                <div align="center">创建时间</div>
            </td>
            <td width="10%" class="td_bg">
                <div align="center">创建人</div>
            </td>
            <td class="td bg" width="15%">
                <div align="center">所属栏目</div>
            </td>
            <td width="15%" class="td_bg">
                <div align="center">文章导读</div>
            </td>
            <td width="10%" class="td_bg">
                <div align="center">语言</div>
            </td>
            <td class="td_bg" width="10%">
                <div align="center">编辑</div>
            </td>
            <td class="td bg" width="10%">
                <div align="center">删除</div>
            </td>
        </tr>
        <s:if test="#request.msgs.size()!=0">
            <s:iterator value="#request.msgs" var="msg">
                <tr>
                    <td height="25" style="color: #993333;">
                        <s:if test="#msg.name.trim().length()>30">
                            <s:property
value="%{#msg.name.substring(0,30)}"/>……                         </s:if>
                        <s:else>
                            <s:property value="#msg.name"/>
                        </s:else>
                    </td>
                    <td><s:date name="#msg.createTime"
format="yyyy-MM-dd"/></td>
                    <td><s:property value="#msg.createUser"/></td>
                    <td class="td_bg">
                        <s:if test="#msg.msgType==2">企业简介</s:if>
```

```
                        <s:elseif test="#msg.msgType==3">联系我们</s:elseif>
                        <s:elseif test="#msg.msgType==4">培训招生</s:elseif>
                        <s:elseif test="#msg.msgType==5">在线视频</s:elseif>
                    </td>
                    <td>
                        <s:if test="#msg.headMsg.trim().length()>30">
                            <s:property
value="%{#msg.headMsg.substring(0,30)}" />…
                        </s:if>
                        <s:else>
                            <s:property value="#msg.headMsg"/>
                        </s:else>
                    </td>
                    <td class="td_bg">
                        <s:if test="#msg.langType.equals(\"zh\")">中文</s:if>
                        <s:else>英文</s:else>
                    </td>
                    <td width="10%">【编辑】</td>
                    <td width="10%">【删除】</td>
                </tr>
            </s:iterator>
        </s:if>
        <s:else>
            <tr>
                <td colspan="8" align="center" height="25px">暂无数据</td>
            </tr>
        </s:else>
        <s:if test="#request.msgs.size()!=0">
            <tr>
                <th class="bg_tr" align="right" colspan="8" height="25">
                    <s:push value="#request.session.pageBean">共 <span
class="red"><s:property value="totalCount" /></span>条记录 当前第<span
class="red"><s:property value="nowpage+1" /></span>/<s:property
value="totalpage" />页 每页<s:property value="pagesize" />条数据
                        </s:push>
                        <s:if test="#request.sendpage==null">
                            <s:a
href="message/message.action?%{#request.sign}&mark=f&number=0">首页</s:a>
                            <s:a
href="message/message.action?%{#request.sign}&mark=p&number=0">上一页</s:a>
                            <s:a
href="message/message.action?%{#request.sign}&mark=n&number=0">下一页</s:a>
                            <s:a
href="message/message.action?%{#request.sign}&mark=l&number=0">尾页</s:a>
                        </s:if>
                        <s:else>
                            <s:a
href="message/message.action?sign=search&%{#request.sign}&mark=f&number=0">
首页</s:a>
```

```
                            <s:a
href="message/message.action?sign=search&%{#request.sign}&mark=p&number=0">
上一页</s:a>
                            <s:a
href="message/message.action?sign=search&%{#request.sign}&mark=n&number=0">
下一页</s:a>
                            <s:a
href="message/message.action?sign=search&%{#request.sign}&mark=l&number=0">
尾页</s:a>
                        </s:else>
                    </th>
                </tr>
            </s:if>
    </table>
```

(4) 在本页面中添加 JavaScript，代码如下所示。

```
<script type="text/javascript">
    function search(){
        var tiao=document.getElementById("sign").value;
        if(tiao==0){
            document.getElementById("divmsg").style.display="none";
            document.getElementById("divlang").style.display="none";
            document.getElementById("divsign").innerHTML="请选择您要查询的条件!";
        }
        else{
            if(tiao=="all"){
                window.location.href="message/message.action";
            }
            else if(tiao=="msgType"){
                document.getElementById("divmsg").style.display="inline";
                document.getElementById("divlang").style.display="none";
            }else{
                document.getElementById("divmsg").style.display="none";
                document.getElementById("divlang").style.display="inline";
            }
            document.getElementById("divsign").innerHTML="";
        }
    }
    function checkmsg(msgString){
        if(msgString==0){
            document.getElementById("divsign").innerHTML="请选择所属栏目名称! ";
        }
        else{

    window.location.href="message/message.action?sign=search&msgString="+ms
gString;
        }
    }
    function checklang(langString){
        if(langString==0){
```

```
                document.getElementById("divsign").innerHTML="请选择语言类型！";
            }else{

        window.location.href="message/message.action?sign=search&langString="+l
angString;
            }
        }
</script>
```

下面来测试一下运行效果。单击后台管理系统首页信息管理模块中的信息管理模块，出现如图 14-18 所示的界面。

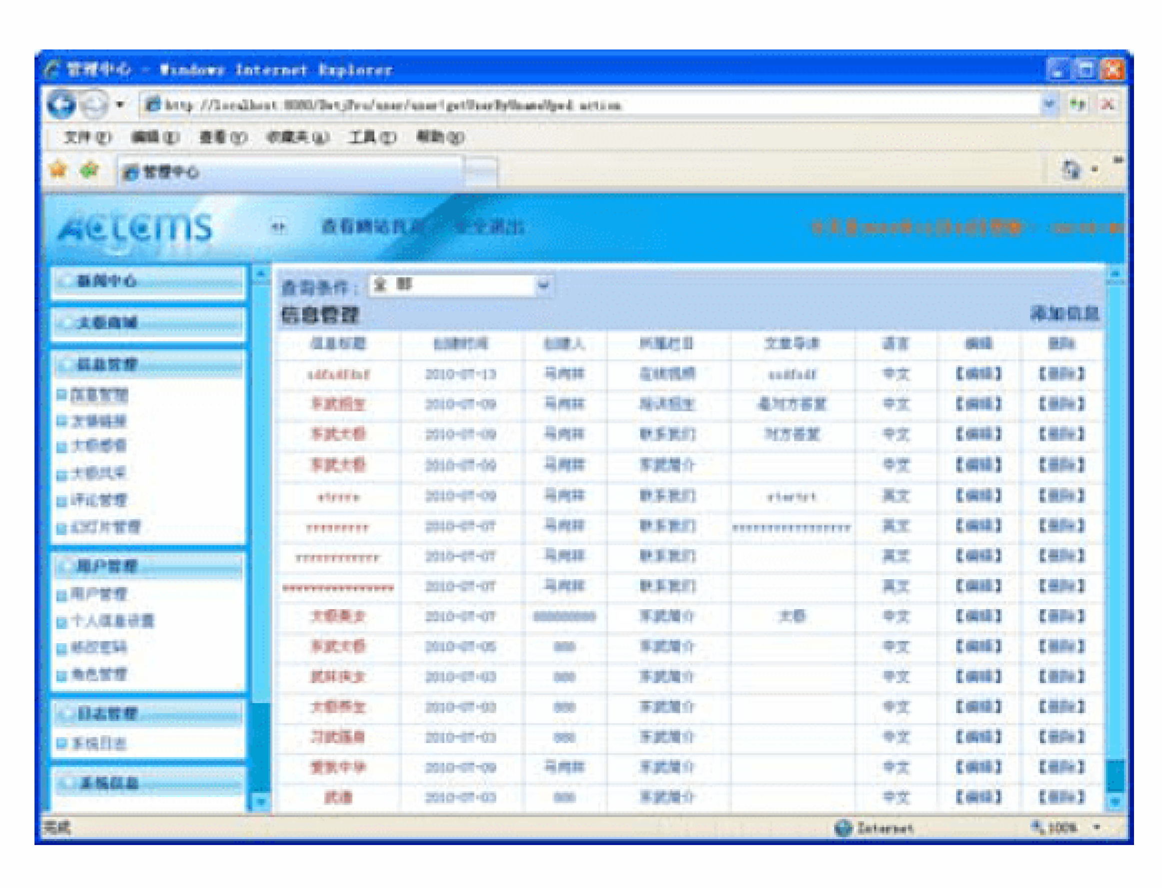


图 14-18　信息管理初始化界面

选择左上角“查询条件”下拉列表框中的“所属栏目”选项，执行 MessageAction 中的 execute()方法，查询出 Messages 表中的企业简介、培训招生、在线视频和联系我们 4 个栏目下的所有信息，并分页显示，如图 14-19 所示。再次选择“--请选择所属栏目—”下拉列表框中的“企业简介”选项，还是执行 MessageAction 类中的 execute()方法，查询条件为“msgType=2”的所有信息，即企业简介信息，如图 14-20 所示。



图 14-19　查询条件为所属栏目



图 14-20　查询条件为企业简介

上面演示了查询条件为“所属栏目”的信息管理界面效果实例，还可以根据语言类型来进行查询，这里不再演示。至于信息的添加、更新和删除，和上面的实现思路一样，这里不再做详细讲解。

14.7.2 友情链接

通过一个网站的友情链接可以转到其他网站，这是一个成功的企业网站中不可或缺的一个模块。

(1) 在 com.dwtj.actions 包下新建 FriendLinkAction 类，继承自 com.opensymphony.xwork2.ActionSupport 类，并重写父类的 execute()方法，获取友情链接信息列表，方法的实现代码如下所示。

```
    //查询出所有的友情链接信息
    public String execute() throws Exception {
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        //调用业务层方法，获取友情链接信息列表
        List<FriendLink> friendList = fLinkService.getLinkList();
        //把获取到的列表信息保存至 HttpServletRequest 对象中
        request.setAttribute("friendList", friendList);
        return "index";
    }
```

(2) 在 struts-config 文件夹下新建 friendlink.xml 文件，并配置 FriendLinkAction 类及“index”的结果页面，配置如下。

```
<package name="friendlink" namespace="/friendlink" extends="ma">
        <action name="friendlink" class="com.dwtj.actions.FriendLinkAction">
            <result name="index">/admin/friendlink/index.jsp</result>
        </action>
</package>
```

(3) 在 admin 文件夹下新建 friendlink 文件夹，并在该文件夹下新建 index.jsp 页面，显示友情链接所有数据信息，代码片段如下。

```
<s:if test="#request.friendList.size()!=0">
    <s:iterator value="#request.friendList" var="fl">
        <tr>
            <td height="25"><s:property value="#fl.title"/></td>
            <td class="td_bg">
                <s:if test="#fl.addressUrl.trim().length()>20">
                    <s:property value="%{#fl.addressUrl.substring(0,20)}"
/>…
                </s:if>
                <s:else><s:property value="#fl.addressUrl" /></s:else>
            </td>
            <td>
```

```
                <img src="<s:property value="#fl.imgUrl" />" width="40px"
height="30px">
            </td>
            <td class="td_bg">【编辑】</td>
            <td>【删除】</td>
        </tr>
    </s:iterator>
</s:if>
<s:else>
    <tr>
        <td colspan="5" class="td_bg" height="25px">暂无友情链接信息</td>
    </tr>
</s:else>
```

友情链接模块的运行效果如图 14-21 所示。

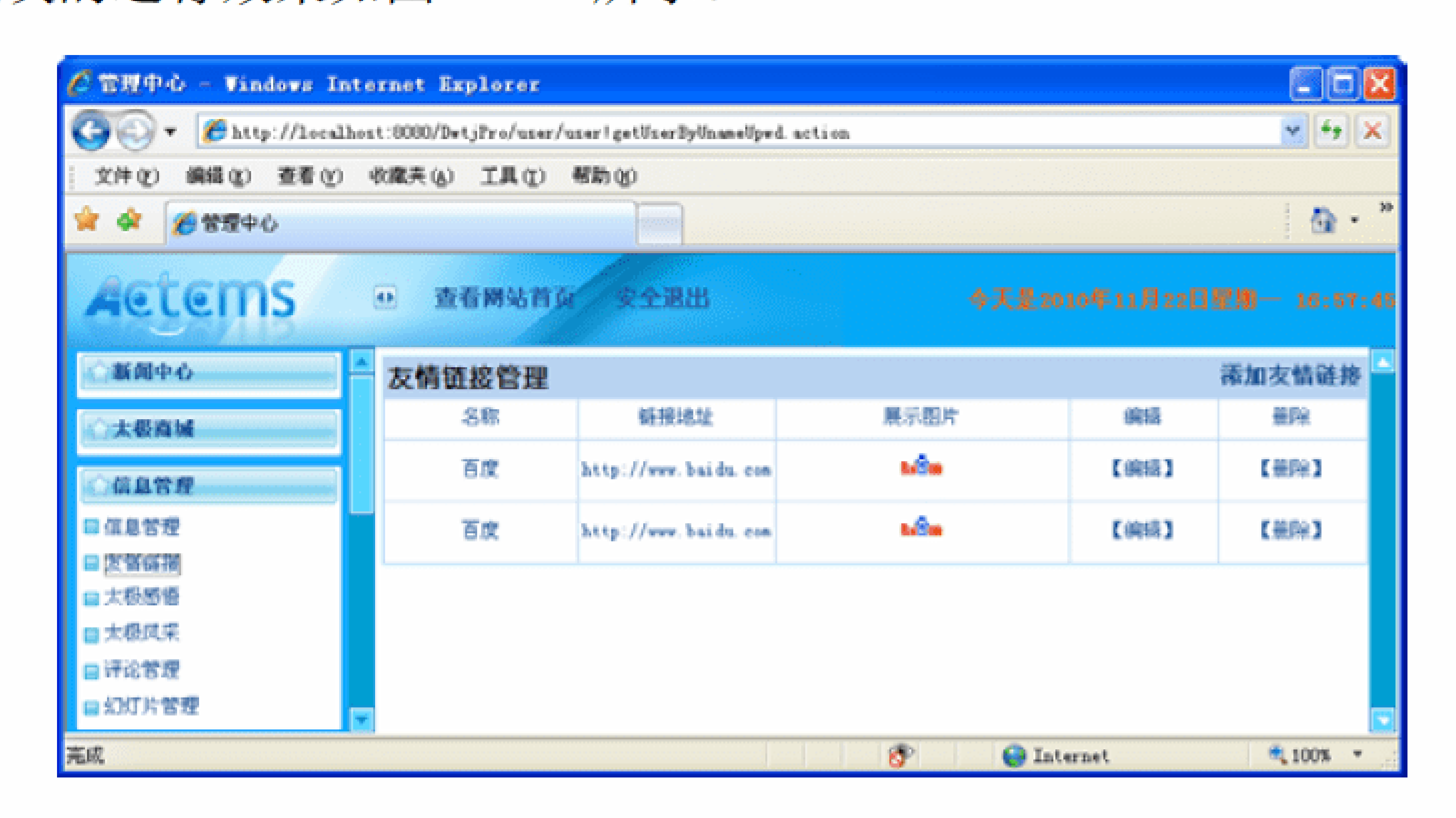


图 14-21　后台模块中的友情链接管理界面

太极感悟、太极风采、评论管理和幻灯片管理的实现，这里不再详细讲解，它们都是独立的表，代码的实现非常简单，读者可以查看本案例中的源码实现代码来做进一步的了解。

14.8　前台展示——在线视频

通过后台的编辑视频，浏览者可在前台在线观看视频内容。由于视频数据比较多，这里采用列表形式显示数据。

14.8.1　获取视频列表信息

(1) 在 MessageAction 类中编辑 getVideo()方法，调用业务层方法获取视频列表，代码如下。

```
    //获取视频列表信息
    public String getVideo(){
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
```

```
        PageBean pageBean = (PageBean) request.getSession().getAttribute(
                "pageBean");
        if (pageBean == null) {
            pageBean = new PageBean();
        }
        String mark = "";
        if (request.getParameter("mark") != null) {
            mark = request.getParameter("mark");
        } else {
            mark = "f";
        }
        int tempNumber = 0;
        if (request.getParameter("number") != null) {
            tempNumber = Integer
                    .parseInt(request.getParameter("number").trim());
        }
        // 调用业务层分页显示数据
        pageBean = msgService.selectVideoAll(PageCtr.PageCtrUpdate(
                pageBean, mark, tempNumber), "");
        request.setAttribute("msgs", pageBean.getResult());
        pageBean.setResult(null);
        request.getSession().setAttribute("sign", "video");//获取标识
        request.getSession().setAttribute("pageBean", pageBean);
        request.getSession().setAttribute("lang", msg.getLangType());
        // 获取语言
        return "newsList";
    }
```

(2) 在 struts-config 文件夹下的 message.xml 文件中配置“newsList”结果页面，配置如下。

```
<result name="newsList">/newList.jsp</result>
```

(3) 在根目录下新建 newList.jsp 页面，遍历视频列表信息，输出视频信息，代码片段如下。

```
<s:if test="#request.msgs.size()!=0">
    <s:iterator value="#request.msgs" var="msg">
            <s:if test="#msg.name.trim().length()>25">
                <s:property value="%{#msg.name.substring(0,25)}" />…
            </s:if>
            <s:else>
                <s:property value="#msg.name" />
            </s:else>
    </s:iterator>
</s:if>
<s:else>
    暂无信息
</s:else>
```

运行程序，视频列表页面如图 14-22 所示。

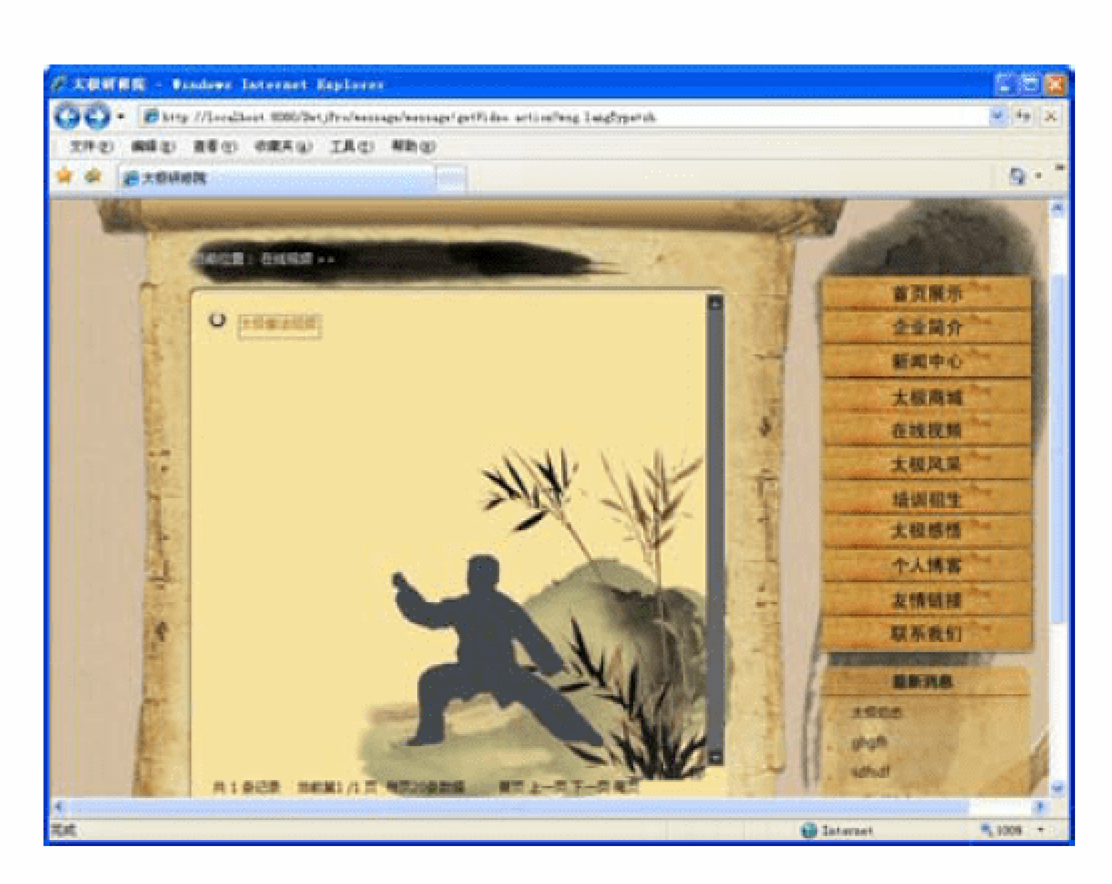


图 14-22　视频列表页面

14.8.2　获取特定的视频信息

单击视频列表中的视频名称，显示特定的视频信息，以供浏览者查看视频内容。

给视频名称设置超链接。

```
<a style="cursor: pointer;" href="message/message!findNewById.action?msg.id=
<s:property value="#msg.id"/>&msg.langType=<s:property
value="#session.lang"/>&sign=<s:property value="#session.sign"/>">
    <s:if test="#msg.name.trim().length()>25">
        <s:property value="%{#msg.name.substring(0,25)}" />…
    </s:if>
    <s:else>
        <s:property value="#msg.name"/>
    </s:else>
</a>
```

从超链接地址来看，这里要访问的是 MessageAction 类中的 findNewById()方法，并将视频 Id 传递给了这个方法，这个方法就是前面 14.4.2 节中讲到的 findNewById()方法，这里不再赘述。

测试一下效果！单击视频列表页面中的视频名称，出现如图 14-23 所示的界面。



图 14-23　显示特定的视频信息

14.9 前台展示——友情链接

单击前台页面中的“友情链接”栏目，显示友情链接数据；单击友情链接信息中的图片，在新窗口中打开连接到的网页。

(1) 在 FriendLinkAction 类中编辑 getLinkUrl()方法，获取友情链接数据，代码如下。

```
    //获取友情链接信息
    public String getLinkUrl(){
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        //调用业务层方法获取友情链接列表
        List<FriendLink> friendList = fLinkService.getLinkList();
        request.setAttribute("friendList", friendList);
        request.getSession().setAttribute("lang",
request.getParameter("lang"));
        return "linkurl";
    }
```

(2) 在 struts-config 文件夹下的 friendlink.xml 文件中配置“linkurl”的结果页面，配置如下。

```
<result name="linkurl">/friend_link.jsp</result>
```

(3) 在项目的根目录下新建 friend_link.jsp 页面，显示友情链接信息，代码片段如下。

```
<s:if test="#request.friendList.size()!=0">
    <s:iterator value="#request.friendList" var="fl">
        <a href="<s:property value="#fl.addressUrl"/>" target="_blank">
            <img src="<s:property value="#fl.imgUrl"/>" alt="<s:property
value="#fl.addressUrl"/>"/>
            <div style="background-color:
black;color:red;width:150px;line-height:
25px;overflow:hidden;float:left;font-size:14px;cursor: pointer;">
                <s:if test="#fl.title.trim().length()>10">
                    <s:property value="%{#fl.title.substring(0,10)}" />…
                </s:if>
                <s:else>
                    <s:property value="#fl.title" />
                </s:else>
            </div>
        </a>
    </s:iterator>
</s:if>
<s:else>暂无友情链接</s:else>
```

(4) 运行程序，单击“友情链接”栏目，跳转至 friend_link.jsp 页面，如图 14-24 所示。

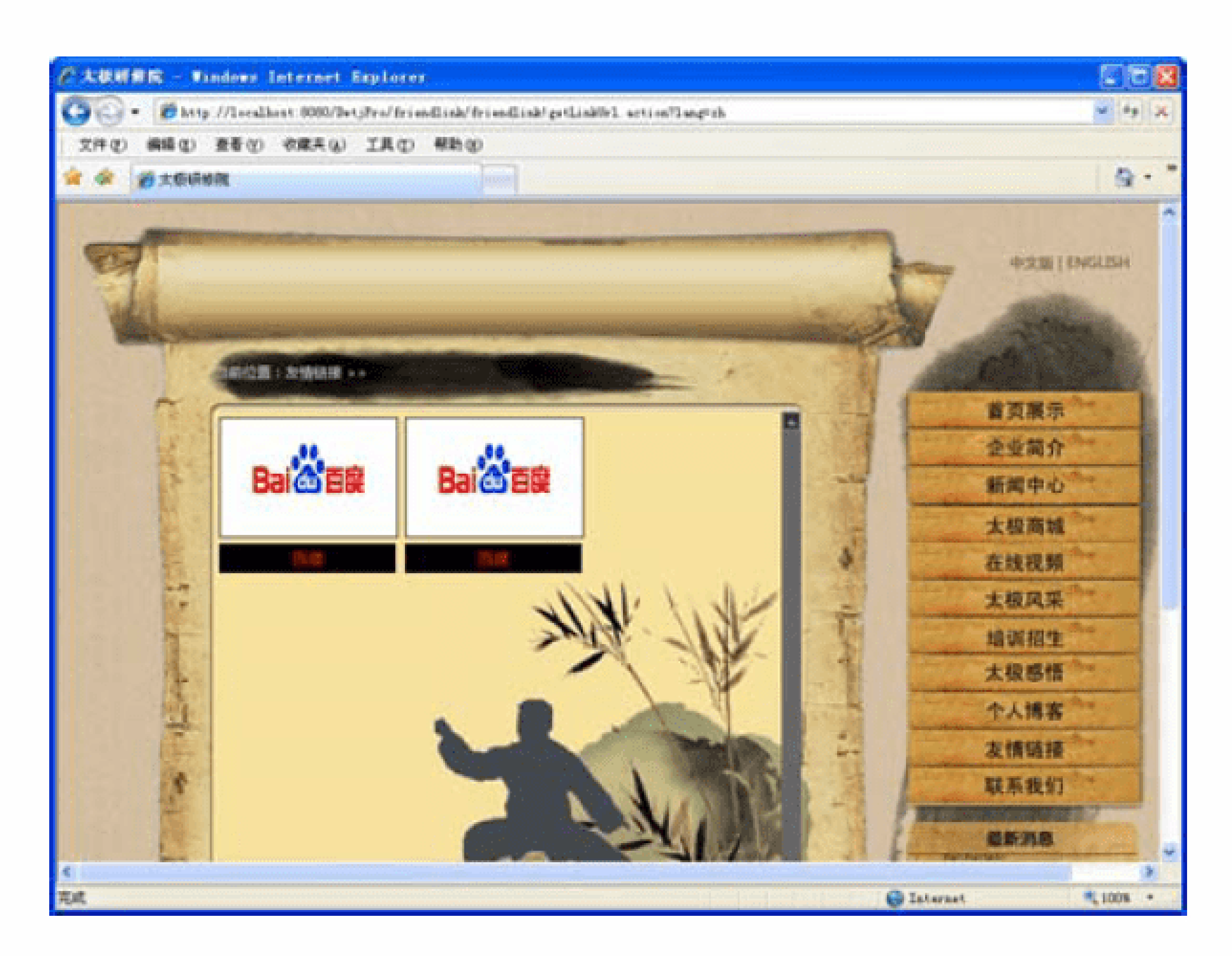


图 14-24　友情链接界面

单击友情链接列表页面中的图片，在新窗口中打开链接的网页，这个超链接是动态加载的，数据表中有一个字段用来存储链接路径。

14.10　总　　结

世上无难事，只怕有心人。当你看到要做一个企业网站，你可能会有一点点的胆怯，觉得做一个网站花费的精力太多，要实现的功能也太多。当你看到项目需求分析时，发现有的功能你不会，事实上，这个网站很简单，没必要恐惧它，相信自己。

由于，本案例中的登录功能和其他系统的功能实现完全一样，这里并没有做详细的介绍。本案例中的用户表里面存放有管理员信息和会员信息，当会员登录后台管理系统时，系统会进行拦截，会员是无法登录到后台管理系统的。当管理员在已经登录的状态下再次登录时，系统也会进行拦截，不让其重复登录。

在案例中，多处用到分页技术。分页技术有多种方法都可以实现，例如 jQuery。

FCKEditor 现在是 Web 开发中最常用的文本编辑器，它的使用非常简单，我也为读者讲解过，如果你想换皮肤或者设置 FCKEditor 的其他属性，都可以通过它的配置文件进行修改。

一个人是否有程序员的潜质，就看自己怎么对待自己的工作了。困难并不可怕，可怕的是没有一颗自信、上进的心。



第 15 章　人力资源管理系统

内容摘要：

在信息社会中，随着电脑与网络技术的日益发达，电子商务空前发展，企业之间的竞争也从有形的市场逐渐转向了虚拟的网络，相应的企业管理也进入了信息化，人力资源管理系统由此诞生。人力资源管理系统是企业管理平台(ERP)的重要组成部分，是为了提高企业的管理水平而设计和开发的管理信息系统。其设计的目标是实现对企业人力资源信息的统一管理，用于支持决策，满足用户及时沟通需要，实现与其他系统的协同。

在本章的人力资源管理系统中，将严格遵循系统开发流程，结合前面章节所学知识，运用 Ajax 技术进行异步提交数据，使用 CSS 控制页面显示样式，客户端页面是 JSP 动态实现，系统整体采用 Struts 2+Hibernate+Spring(SSH)整合的开发模式。

学习目标：

- 掌握系统需求分析的过程。
- 掌握人力资源管理系统的数据库设计。
- 掌握 Struts+Spring+Hibernate 开发模式。
- 掌握 Struts、Spring 配置文件中的配置项。
- 掌握 Hibernate 映射文件的编写。
- 了解业务控制器 Action 的使用。
- 理解 SSH 开发应用。
- 熟悉 MyEclipse 的使用。

15.1 系 统 分 析

系统分析包括系统需求分析和系统可行性分析。从系统需求分析中，主要了解系统的性能需求和功能需求；从系统可行性分析中，主要了解系统的经济效益和系统构建时的技术的可行性。

15.1.1 系统需求分析

人力资源管理系统是一个企事业单位不可少的部分，它对决策者和管理者来说都至关重要，因为人力资源管理系统应该能够为用户提供充足的信息和快捷的查询。但是一直以来，人们都是使用人工的方式来管理文件档案，这种管理方式有很多缺点，如：效率低、保密性差，时间一长，将会产生大量的数据和文件，给用户更新和查找带来了很大的困难。

作为计算机应用的一部分，使用计算机对人事信息进行管理，具有手工管理无法比拟的优点。如：查询方便、检索快、存储量大、保密性好、成本低等。这些优点能够极大地提高企业的人事管理的效率，也是企业的正规化、科学化管理的重要条件。所以，开发一个稳定、高效、功能完善的人力资源管理系统成为了一种必要。

1. 性能需求

一个信息管理系统，第一，要有很好的稳定性和可维护性；第二，要有很好的可扩展性，在不影响基本框架的前提下，增加新的业务逻辑，进行二次开发；第三，要有很好的可移植性，以满足不同系统用户的需求；第四，要尽可能使界面美观、简洁、操作简单。

2. 功能需求

人力资源管理系统需要人力资源的管理不受地点、时间的限制，只要能够上网，通过验证并登录系统，便能够查看人力资源信息，进行人力资源信息的管理和更新。人力资源管理系统具有员工管理、招聘管理、奖惩管理、培训管理、薪资管理及管理员模块。人力资源部门负责添加、修改和删除各类信息。领导可以查询各类信息。员工可以浏览个人的信息。

15.1.2 系统可行性分析

计算机网络作为一种先进的信息传输媒体，有着传送信息块、信息覆盖面广、成本低的优点。因此，很多的企业开始利用网络开展商务活动，可以看到，在企业进行的网上商业活动所产生的效益是多方面的。但是，开发一个基于计算机的系统，都会受到资源和时间的限制。因此，在接受任何一个项目开发任务之前，必须根据提供的时间和资源条件进行可行性分析，以减少项目的开发风险，避免人力、物力和财力的浪费。经济可行性分析和技术可行性分析在很多方面是相互关联的，项目风险越大，开发高质量软件的可行性就越小。

1. 经济可行性

通过网络化的人力资源管理，大大地提高了企业人才的利用率，使之为企业创造了更大的价值。人才利用率的提高，增强了企业的核心竞争力，全面地提升了企业的管理能力，从而使企业适应了信息时代的网络化管理要求。

2. 技术可行性

开发此系统需要的环境有如下几点。

- 操作系统：Windows XP/Windows 2000。
- 数据库：Mysql5.0。
- 开发工具：MyEclipse8.5。
- 服务器：Tomcat。

基于编程开发语言是 Java、JSP，因此，需要开发人员熟练使用 Java、JSP 语言；需要开发人员熟练使用相关数据库的操作，开发人员需要具有一定的数据库开发功底和编程能力。优美的界面设计再加上 Windows 稳定的运行环境的支持和开发人员的过硬技术，从功能和性能上完全可以满足系统的要求，因此，从技术方面此系统是可行的，综合以上两方面，开发此系统是可行的。

15.2　系 统 设 计

系统设计是在系统分析的基础上由抽象到具体的过程，主要目标是将系统分析阶段所提出的反应信息需求的系统逻辑方案，转换成可以实施的基于计算机与通信系统的物理方案，为下一阶段的技术实施提供必要的技术资料，同时须符合系统性、灵活性、可靠性、经济性的要求。

15.2.1　总体设计

系统总体设计是对系统的模块规划、系统功能结构和数据库的总体设计。

1. 模块规划

人力资源管理系统是针对员工信息管理的一个平台，系统的主要功能如下。

- 员工管理：主要包括浏览员工信息、添加员工信息、修改员工信息、员工信息的删除。
- 应聘管理：主要包括应聘人员信息的查看、删除及添加信息入库和修改应聘者信息。
- 培训管理：主要包括培训计划的详细信息浏览、信息删除、添加和修改培训计划。
- 奖惩管理：主要包括浏览奖惩的详细信息、添加信息、删除信息、修改信息。
- 薪资管理：主要包括浏览薪资的详细信息、薪资的修改、添加、删除。

2. 系统功能结构

主要模块结构图如图 15-1 所示。

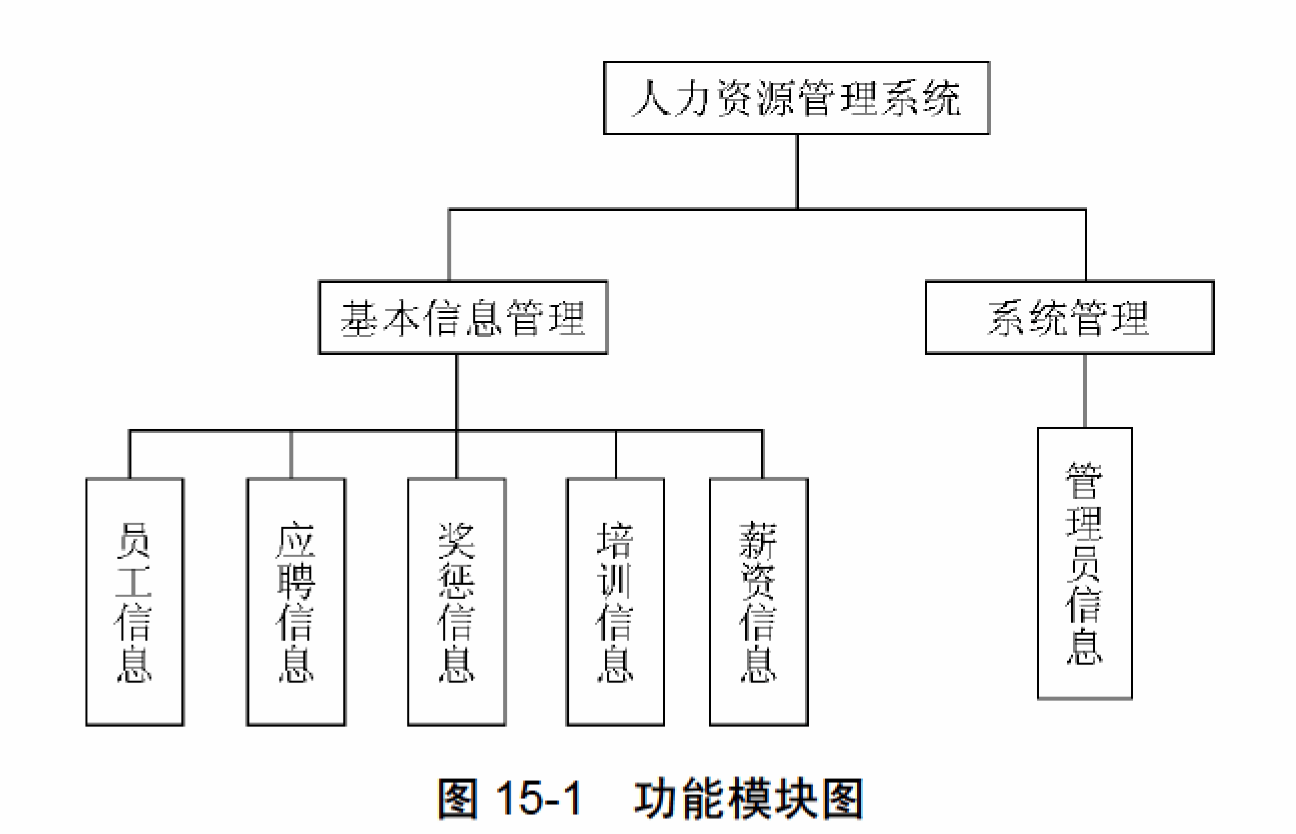


图 15-1　功能模块图

15.2.2　数据库设计

数据库在一个信息管理系统中的地位非常重要，数据库结构设计的好坏将直接对应用系统的效率，实现的效果产生影响。合理的数据库结构设计可以提高数据存储的效率，保证数据的完整和一致。

1. 数据库需求分析

数据库系统应充分了解用户各方面的需求，本系统用户的需求具体体现在各种信息的提供、存储、更新和查询，这要求数据库的结构能充分满足各种信息的输入和输出。收集基本数据、数据结构及数据的处理流程，为后面的具体设计打下基础。

根据系统功能分析和需求总结，考虑到将来功能上的扩展，设计用户信息表、管理员信息表、培训信息表、应聘信息表、奖惩信息表、薪资信息表。

2. 数据库概念设计

根据上面的数据项和数据结构，就可以设计出能够满足用户需求的各种实体，为以后的逻辑结构设计打下基础。因此，本系统的实体有：管理员实体、员工实体、应聘实体、奖惩实体、培训实体、薪资实体。

3. 数据库逻辑结构设计

本系统选用 MySQL 数据库，在 MySQL 数据库中，为本人力资源管理系统建立一个数据库 pmanager。根据数据库的概念设计，确定本系统需要 6 个表，分别是：管理员信息表、员工信息表、应聘信息表、奖惩信息表、培训信息表、薪资信息表。由于本系统的持久化层使用 Hibernate 技术，Hibernate 会在数据库中自动映射生成数据表，所以不需要手动建立。在这里为了让各位读者更加明白，简要介绍一下各个表的结构。

1)　管理员信息表

管理员表用来保存管理员的基本信息，管理员表中主要有管理员 id、用户名、密码三个字段。管理员信息如表 15-1 所示。

表 15-1　管理员表admin

字段名称	含　义	类　型	约　束
Id	管理员 id	Int	主键
Adminname	用户名	Varchar	非空
Password	密码	Varchar	非空

2．员工信息表

员工表用来保存员工的基本信息，员工表中主要有员工 id、员工姓名、密码、性别、电话、地址、生日、简介 8 个字段。员工表信息如表 15-2 所示。

表 15-2　员工信息表employee

字段名称	含　义	类　型	约　束
Id	员工 id	Int	主键
Name	员工姓名	Varchar	非空
Password	员工密码	Varchar	非空
Sex	员工性别	Varchar	可空
Tel	员工电话	Varchar	可空
Address	员工地址	Varchar	可空
Birthday	出生日期	Date	可空
Content	员工简介	Varchar	可空

3．应聘信息表

应聘信息表用来保存应聘的人员的基本信息，应聘表中主要有应聘者 id、名称、性别、年龄、应聘职位、专业、工作经验、学历、学校、电话、邮箱、简介 12 个字段。应聘表信息如表 15-3 所示。

表 15-3　应聘信息表apply

字段名称	含　义	类　型	约　束
Id	应聘者 id	Int	主键
Name	名称	Varchar	可空
Sex	性别	Varchar	可空
Age	年龄	Varchar	可空
Post	应聘职位	Varchar	可空
Professional	专业	Varchar	可空
Experience	经验	Varchar	可空
Education	学历	Varchar	可空
School	学校	Varchar	可空
Tel	电话	Varchar	可空
Email	邮箱	Varchar	可空
Content	简介	Varchar	可空

4. 奖惩信息表

奖惩信息表用来保存奖惩的基本信息，奖惩表中主要有奖惩 id、奖惩名字、奖惩原因、奖惩说明 4 个字段。奖惩表信息如表 15-4 所示。

表 15-4 奖惩信息表incentives

字段名称	含 义	类 型	约 束
Id	奖惩 id	Int	主键
Jname	奖惩名字	Varchar	可空
Reason	奖惩原因	Varchar	可空
Jcontent	奖惩说明	Varchar	可空

5. 培训信息表

培训信息表用来保存培训的基本信息，培训表中主要有培训 id、培训名称、培训目的、培训开始时间、培训结束时间、培训讲师、培训人员、培训内容 8 个字段。培训表信息如表 15-5 所示。

表 15-5 培训信息表train

字段名称	含 义	类 型	约 束
Id	培训 id	Int	主键
Trainname	培训名称	Varchar	可空
Target	培训目的	Varchar	可空
Starttime	开始时间	Date	可空
Stoptime	结束时间	Date	可空
Teacher	讲师	Varchar	可空
Student	培训人员	Varchar	可空
Content	培训内容	Varchar	可空

6. 薪资信息表

薪资信息表用来保存薪资的基本信息，薪资表中主要有薪资 id、员工姓名、基本工资、食补、房补、全勤奖、罚款、发放时间、总计 9 个字段。薪资表信息如表 15-6 所示。

表 15-6 薪资信息表pay

字段名称	含 义	类 型	约 束
Id	薪资 id	Int	主键
Name	员工姓名	Varchar	非空
Basic	基本工资	Float	非空
Eat	食补	Float	可空
House	房补	Float	可空
Duty	全勤奖	Float	可空
Othor	罚款	Float	可空
Granttime	发放时间	Date	可空
Total	总计	Float	可空



在每个数据表中，都设置一个自增列。一方面让每条记录信息都有唯一标识符；另一方面可以作为外键，关联到其他表。

15.3　系统运行和开发环境的搭建

本章从配置项目的开发环境，到整个项目的完成，都是以 SSH 整合的运用进行项目开发的过程，所以只要读者用心就一定能够成功。表 15-7 所示是搭建开发环境所需的软件，下载后进行安装、配置环境。

表 15-7　环境配置及开发工具软件

软　件	版　本	下载地址
JDK	1.6.0	http://java.sun.com
Tomcat	6.0	http://tomcat.apache.org
MySQL	5.0	http://dev.mysql.com/
MyEclipse	7.0	http://downloads.myeclipseide.com/

在开发工具 MyEclipse 中，创建一个名为 Pmanager 的 Web 工程，搭建好运行环境。本人力资源管理系统的目录结构如图 15-2 所示。

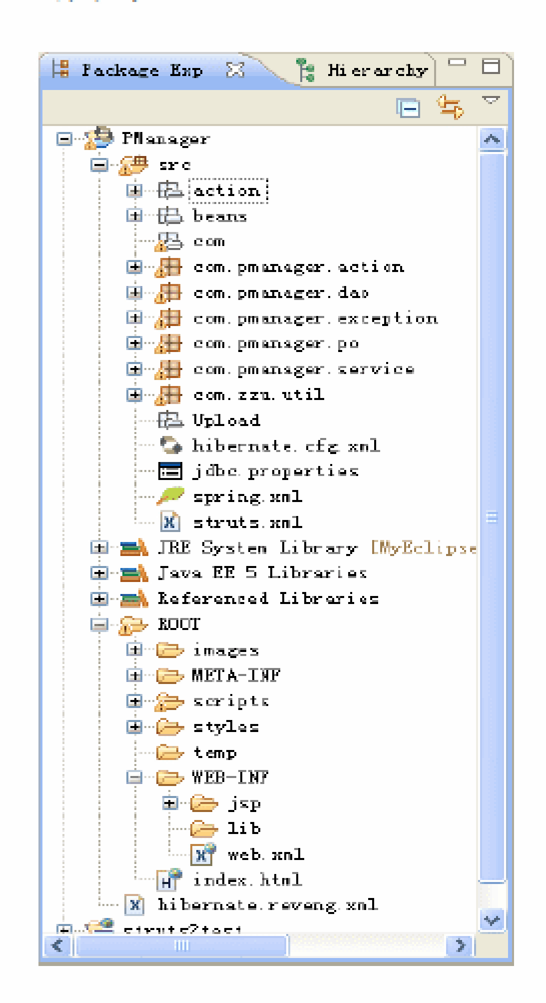


图 15-2　人力资源管理系统目录结构图

在图 15-2 所示的目录结构中/action 中存放各个 action 的配置文件，/bean 中存放各个 bean 文件的映射配置文件，com/pmanager/action 中存放各个模块的 action，com/pmanager/dao 中存放各个数据库访问类 dao，com/pmanager/service 中存放各个模块的代理类 service，com/pmanager/po 中存放各个表的实体类及实体类的 hibernate 映射文件，com./zzu/util 中存放各种使用工具类，ROOT 目录下保存的是 JSP 文件和相应的 CSS 文件和图片，WebRoot/WEB-INF 目录下是 web.xml 配置文件，src 目录中存放 jdbc 资源文件和 SSH 的 Struts.xml、applicationContext.xml、Hibernate.cfg.xml 配置文件。以下是对各个配置文件的介绍。

15.3.1　web.xml配置文件

web.xml 是 web 应用中加载有关 Servlet 信息的重要配置文件，起着初始化 servlet、filter 等 web 程序的作用。

每一个 xml 文件都有定义书写规范的 schema 文件，web.xml 所对应的 xml Schema 文件中定义了多少种标签元素，web.xml 中就可以出现它所定义的标签元素，也就具备哪些特定的功能。web.xml 的模式文件是由 Sun 公司定义的，每个 web.xml 文件的根元素为<web-app>中，必须标明这个 web.xml 使用的是哪个模式文件。详细配置如下代码。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:/spring.xml</param-value>
  </context-param>
  <listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</list
ener-class>
  </listener>
  <filter>
    <filter-name>Struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecu
teFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>Struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

15.3.2　struts.xml配置文件

Struts 2 的核心配置文件是缺省的 strutsl.xml,这个文件也是 struts 2 框架主动加载的文件，在这个文件中可以定义自己的一些 action、interceptor、package 等。Package 通常继承 struts-default 包。Struts 文件可以放入 jar 中，并自动插入应用程序，这样每个模块可以包含自己的配置文件并自动配置。详细配置如下代码。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 2.1//EN" "http://struts.apache.org/dtds/struts-2.1.dtd">
<struts>
    <constant name="struts.ognl.allowStaticMethodAccess" value="true" />
    <!-- 更改 struts2 的标签为简单样式 -->
    <constant name="struts.ui.theme" value="simple" />
    <constant name="struts.ui.emplateDir" value="template" />
    <constant name="struts.ui.templateSuffix" value="ftl" />
    <constant name="struts.enable.SlashesInActionNames" value="true" />
    <constant name="struts.mapper.alwaysSelectFullNamespace" value="false" />
    <!-- 结合 spring 插件 -->
    <constant name="struts.objectFactory" value="spring"></constant>
    <constant name="struts.locale" value="en_utf-8"></constant>
  <!-- 公用的 jsp 映射  -->
    <package name="jsp" extends="struts-default" namespace="/jsp">
   <action name="**">
     <result name="success">/WEB-INF/jsp/{1}.jsp</result>
   </action>
  </package>
  <package name="default" extends="struts-default">
    <global-results>
      <result name="error">/WEB-INF/jsp/error.jsp</result>
    </global-results>
  </package>
    <include file="action/Main.xml"/>
    <include file="action/EmpManager.xml"/>
    <include file="action/AdminManager.xml"/>
    <include file="action/RecruitManager.xml"/>
    <include file="action/IncentivesManager.xml"/>
    <include file="action/TrainManager.xml"/>
    <include file="action/PayManager.xml"/>
</struts>
```

以下是其中一个模块的 struts 配置文件的代码。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 2.1//EN" "http://struts.apache.org/dtds/struts-2.1.dtd">
<struts>
   <package name="AdminManager" extends="default" namespace="/admin">
      <action name="main" class="adminManager" method="main">
         <result name="success">/WEB-INF/jsp/Admin/main.jsp</result>
      </action>
      <action name="list" class="adminManager"  method="list"/>
       <action name="add" class="adminManager" method="add">
         <result name="success" type="redirect">main</result>
       </action>
       <action name="modify" class="adminManager" method="modify">
         <result name="success" type="redirect">main</result>
```

```
        </action>
       <action name="load" class="adminManager" method="load" />

    </package>
</struts>
```

15.3.3　hibernate.cfg.xml配置文件

Hibernate 主要用于配置数据库连接、事务管理，以及指定 Hibernate 本身的配置信息和 Hibernate 映射文件信息。Hibernate 配置文件默认以 hibernate.properties 或者 hibernate.cfg.xml 命名，常用的是 XML 格式的配置文件，这个配置文件应该位于应用的 classpath 中。具体配置代码如下所示。

```
<session-factory>
    <property name="dialect">
        org.hibernate.dialect.MySQLDialect
    </property>
    <property name="connection.url">
        jdbc:mysql://localhost:3306/pmanager
    </property>
    <property name="connection.username">root</property>
    <property name="connection.password">root</property>
    <property name="connection.driver class">
        com.mysql.jdbc.Driver
    </property>
    <property name="myeclipse.connection.profile">mysql</property>
    <property name="hibernate.show_sql">true</property>

    <mapping resource="com/pmanager/po/Admin.hbm.xml" />
    <mapping resource="com/pmanager/po/Company.hbm.xml" />
    <mapping resource="com/pmanager/po/Employee.hbm.xml" />
    <mapping resource="com/pmanager/po/Incentives.hbm.xml" />
    <mapping resource="com/pmanager/po/Pay.hbm.xml" />
    <mapping resource="com/pmanager/po/Recruitment.hbm.xml" />
    <mapping resource="com/pmanager/po/Train.hbm.xml" />
</session-factory>
</hibernate-configuration>
```

15.3.4　applicationContext.xml配置文件

Spring 对 Hibernate 提供了良好的支持，Hibernate 的配置信息完全可以在 Spring 的配置文件中配置。Hibernate 和 Spring 集成使用，可以不需要 hibernate.cfg.xml 文件。

编辑 ROOT\WEB-INF 目录下的 applicationContext.xml，配置 Hibernate 的相关信息。配置代码如下。

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!--  - Application context definition for "springapp" DispatcherServlet.  -->
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
        http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">
<context:property-placeholder location="classpath:jdbc.properties" />
<bean id="datasource"
       class="org.apache.commons.dbcp.BasicDataSource"
       destroy-method="close">
   <property name="driverClassName" value="${jdbc.driverClassName}" />
   <property name="url" value="${jdbc.url}" />
   <property name="username" value="${jdbc.username}" />
   <property name="password" value="${jdbc.password}" />
  </bean>
  <!-- Hibernate SessionFactory -->
  <bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
   <property name="dataSource" ref="datasource"/>
   <property name="configLocations"
value="classpath:/hibernate.cfg.xml"></property>
  </bean>
  <!-- Transaction manager for a single JDBC DataSource -->
  <bean id="transactionManager"

class="org.springframework.orm.hibernate3.HibernateTransactionManager">
   <property name="sessionFactory" ref="sessionFactory" />
  </bean>
  <tx:annotation-driven transaction-manager="transactionManager" />

  <bean id="hibernateTemplate"
class="org.springframework.orm.hibernate3.HibernateTemplate">
   <property name="sessionFactory" ref="sessionFactory" />
  </bean>
  <!-- 各个模块 Bean 声明的导入 -->

  <import resource="classpath:/beans/Main.xml"/>
  <import resource="classpath:/beans/Emp.xml"/>
  <import resource="classpath:/beans/Admin.xml"/>
  <import resource="classpath:/beans/Recruit.xml"/>
  <import resource="classpath:/beans/Incentives.xml"/>
  <import resource="classpath:/beans/Train.xml"/>
  <import resource="classpath:/beans/Pay.xml"/>
</beans>
```

15.4　系统的实现

对于企业来说，一个人力资源管理系统，其简洁明了的页面风格和严谨的逻辑结构是必不可少的。

由于此项目功能的实现太多，我不再一一介绍，读者可以根据本章源码来做具体的分析，这里只讲解典型的模块功能的实现。

首先，系统进入主界面，可以看到人力资源管理系统主要包括基本信息管理和系统管理两大模块，其中基本信息管理模块包括员工信息管理、应聘信息管理、奖惩信息管理、培训信息管理和薪资信息管理五大管理模块，系统管理模块中有管理员信息管理模块，分别点击相应的标题可以进入各部分界面，如图 15-3 所示。



图 15-3　人力资源管理系统主界面

15.4.1　管理员模块——代码开发步骤

管理员模块主要包括管理员列表、添加管理员、修改管理员和删除管理员四部分。单击“管理员信息管理”和“管理员列表”按钮可以看到管理员信息的表单，如图 15-4 所示。



图 15-4　管理员信息表单

以上所看到的浏览信息的分页显示，在整个项目中都要用到，在这一节里将详细地讲解一个功能模块的具体实现方法，读者可以模仿以下模块自己动手实现。以下是具体的实现步骤。

(1) 在项目中新建 com.pmanger.po 包，在该包下用 hibernate 的数据库反向工程生成 admin 实体类和该类的实体类映射。首先在 MyEclipse 里的右上角打开 MyEclipse database Explorer 界面，新建一个与 MySQL 数据库的链接，然后找到 admin 表，右键单击 Hibernate Reverse Engineering 反向工程生成 admin 实体类和实体类映射。代码如下所示。

```
admin 实体类代码:
package com.pmanager.po;
/**
 * Admin entity. @author MyEclipse Persistence Tools
 */
public class Admin implements java.io.Serializable {
    // Fields
    private Integer id;
    private String adminname;
    private String password;
    // Constructors
    /** default constructor */
    public Admin() {
    }
    /** minimal constructor */
    public Admin(Integer id) {
        this.id = id;
    }
    /** full constructor */
    public Admin(Integer id, String adminname, String password) {
        this.id = id;
        this.adminname = adminname;
        this.password = password;
    }
    // Property accessors
    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }
    public String getAdminname() {
        return this.adminname;
    }
    public void setAdminname(String adminname) {
        this.adminname = adminname;
    }
    public String getPassword() {
        return this.password;
    }
```

```
    public void setPassword(String password) {
        this.password = password;
    }
}
```

下面是随反向工程一起生成的 admin 的实体类映射文件 Admin.hbm.xml。

```
admin 映射文件代码:
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!--
   Mapping file autogenerated by MyEclipse Persistence Tools
-->
<hibernate-mapping>
   <class name="com.pmanager.po.Admin" table="admin" catalog="pmanager">
      <id name="id" type="java.lang.Integer">
         <column name="id" />
         <generator class="assigned" />
      </id>
      <property name="adminname" type="java.lang.String">
         <column name="adminname" length="30">
            <comment>管理员</comment>
         </column>
      </property>
      <property name="password" type="java.lang.String">
         <column name="password" length="50">
            <comment>密码</comment>
         </column>
      </property>
   </class>
</hibernate-mapping>
```

(2) 创建 DAO 层架构。在项目中新建 com.pmanager.dao，该包下存放持久层的所有接口，在该包下新建 AdminDao 接口，负责与持久化对象交互，封装了数据的增、删、改、查等操作，内容如下。

```
package com.pmanager.dao;
import com.pmanager.exception.ModelException;
import com.pmanager.po.Admin;
import com.zzu.util.PageList;

public interface AdminDao {
    /**
      * 加载管理员信息列表(按 admin 参数进行模糊匹配)
      * @param admin 模糊匹配对象
      * @return 管理员列表
      * @throws ModelException 任何可能的业务逻辑异常,异常包含异常消息和错误代码
      */
```

```
    public PageList<Admin> list(Admin admin,int skip,int size) throws
ModelException;
    /**
     * 添加管理员信息
     * @param admin  管理员信息
     * @return 管理员信息
     * @throws ModelException 任何可能的业务逻辑异常,异常包含异常消息和错误代码
     */
    public Admin add(Admin admin) throws ModelException;
    /**
     * 加载管理员信息
     * @param id 加载管理员信息
     * @return id
     * @throws ModelException 任何可能的业务逻辑异常,异常包含异常消息和错误代码
     */
    public Admin load(int id) throws ModelException;
    /**
     * 修改管理员信息
     * @param admin 管理员信息
     * @throws ModelException 任何可能的业务逻辑异常,异常包含异常消息和错误代码
     */
    public void modify(Admin admin) throws ModelException;
}
```

(3)　创建 DAO 层实现类。在项目中的 com.pmanager.dao 包中，新建 AdminDao 接口的实现类 AdminDaoImpl，实现接口中的方法，内容如下。

```
package com.pmanager.dao;

import java.sql.SQLException;
import org.hibernate.Criteria;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.criterion.MatchMode;
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Projections;
import org.hibernate.criterion.Restrictions;
import org.springframework.orm.hibernate3.HibernateCallback;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
import com.pmanager.exception.ModelException;
import com.pmanager.po.Admin;
import com.zzu.util.PageList;
public class AdminDaoImpl extends HibernateDaoSupport implements AdminDao  {
    @SuppressWarnings("unchecked")
    public PageList<Admin> list(final Admin admin, final int skip,final int
size)
          throws ModelException {
         try{
```

```
            PageList<Admin> list = (PageList<Admin>)
this.getHibernateTemplate().execute(new HibernateCallback(){

            public Object doInHibernate(Session session)//获取 session 对象
                    throws HibernateException, SQLException {
                Criteria query = session.createCriteria(Admin.class);
                //查询 admin 表
                //组合条件
                if(admin.getAdminname()!=null){
                    query.add(Restrictions.like("adminname",
admin.getAdminname(),MatchMode.ANYWHERE));
                }
                //统计行数
                query.setProjection(Projections.rowCount());
                Long count = (Long) query.uniqueResult();
                //查询结果
                query.setProjection(null);
                query.addOrder(Order.asc("id"));
                query.setFirstResult(skip);
                query.setMaxResults(size);
                PageList<Admin> list = new PageList(count.intValue());
                list.addAll(query.list());
                return list;
            }});
            return list;
        }catch(Exception e){
         e.printStackTrace();
         throw new ModelException(1,e.getMessage());
        }
    }
    public Admin add(Admin admin) throws ModelException {
     try{
         this.getHibernateTemplate().save(admin);
         this.getHibernateTemplate().flush();

         return admin;
     }catch(Exception e){
         e.printStackTrace();
         throw new ModelException(1,e.getMessage());
        }
    }
    public Admin load(int id) throws ModelException {

         try{
             return (Admin) this.getHibernateTemplate().get(Admin.class, id);
         }catch(Exception e){
         e.printStackTrace();
         throw new ModelException(1,e.getMessage());
        }
```

```
    }
    public void modify(Admin admin) throws ModelException {
        Admin adm = (Admin) this.getHibernateTemplate().get(Admin.class,
admin.getId());
       try{
        adm.setAdminname(admin.getAdminname());
        adm.setPassword(admin.getPassword());
         this.getHibernateTemplate().saveOrUpdate(adm);
       }catch(Exception e){
        e.printStackTrace();
        throw new ModelException(1,e.getMessage());
       }
    }
}
```

(4) 创建业务层接口。在项目中新建 com.pmanager.service 包，在该报下新建 AdminService 接口，编写需要实现的方法，代码如下。

```
public interface AdminService {
/**
       * 加载管理员信息列表(按 admin 参数进行模糊匹配)
       * @param admin 模糊匹配对象
       * @return 管理员列表
       * @throws ModelException 任何可能的业务逻辑异常,异常包含异常消息和错误代码
       */
      public PageList<Admin> list(Admin admin,int skip,int size) throws
ModelException;
      /**
       * 添加管理员信息
       * @param admin   管理员信息
       * @return 管理员信息
       * @throws ModelException 任何可能的业务逻辑异常,异常包含异常消息和错误代码
       */
      public Admin add(Admin admin) throws ModelException;
      /**
       * 加载管理员信息
       * @param id 加载管理员信息
       * @return id
       * @throws ModelException 任何可能的业务逻辑异常,异常包含异常消息和错误代码
       */
      public Admin load(int id) throws ModelException;
      /**
       * 修改管理员信息
       * @param admin 管理员信息
       * @throws ModelException 任何可能的业务逻辑异常,异常包含异常消息和错误代码
       */
      public void modify(Admin admin) throws ModelException;
}
```

(5) 编写业务层实现类 AdminServiceImpl，实现上面接口中的 4 个方法，代码如下。

```
public class AdminServiceImpl implements AdminService {
  private AdminDao adminDao = null;//该模块对应的 DAO

public void setAdminDao(AdminDao adminDao) {//设置到的 set 方法
    this.adminDao = adminDao;
}
//添加信息
public Admin add(Admin admin) throws ModelException {
    int s = 10000000;
    int sid = (int) (System.currentTimeMillis()%1000000);
    admin.setId(new Integer(s+sid));
    adminDao.add(admin);
    return null;
}
//浏览信息
public PageList<Admin> list(Admin admin, int skip, int size)
        throws ModelException {

    return adminDao.list(admin, skip, size);
}
//加载信息
public Admin load(int id) throws ModelException {

    return adminDao.load(id);
}
//修改信息
public void modify(Admin admin) throws ModelException {

    adminDao.modify(admin);

}
//过滤查询分页功能
    private int rows = 10;
    private int page = 1;
    private Admin like = new Admin();
    public Admin getLike() {
        return like;
    }

    public void setLike(Admin like) {
        this.like = like;
    }

    public void setRows(int rows) {
        this.rows = rows;
    }
```

```
    public void setPage(int page) {
        this.page = page;
    }

    public void list() {
        try {
            JSONObject root = new JSONObject();
            PageList<Admin> list = adminService.list(like, (page - 1) * rows,
                    rows);
            int count = list.getRows();
            root.put("page", page);
            root.put("total", count / rows + ((count % rows) > 0 ? 1 : 0));
            root.put("records", count);
            JSONArray rows = new JSONArray();
            for (Admin s : list) {
                rows.add(s, JsonUtil.config());
            }
            root.put("rows", rows);

    ServletActionContext.getResponse().setCharacterEncoding("utf-8");
            root.write(ServletActionContext.getResponse().getWriter());
        } catch (Throwable e) {
            e.printStackTrace();
        }
}
```

(6) 编写 action 类。在项目中新建 com.pmanager.action 包，在该包中新建 admin 的 action 类 AdminManager，继承自 com.opensymphony.xwork2.ActionSupport 类，在这里编写分页显示。代码如下。

```
@SuppressWarnings("serial")
public class AdminManager extends ActionSupport {

  private AdminService adminService = null;//注入 service
  private Admin admin = null;//注入 admin
  //实现 Get 和 Set 方法
  public Admin getAdmin() {
        return admin;
    }

    public void setAdmin(Admin admin) {
        this.admin = admin;
    }

    public void setAdminService(AdminService adminService) {
        this.adminService = adminService;
    }
  /**
   *  显示员工主界面
```

```
     * @return  success
     */
    public String main(){
        return "success";
    }
    //过滤查询功能及分页显示功能
    private int rows = 10;每页的行数
    private int page = 1;页数
    private Admin like = new Admin();前台用于使用的 like 字段
    public Admin getLike() {
        return like;
    }

    public void setLike(Admin like) {
        this.like = like;
    }

    public void setRows(int rows) {
        this.rows = rows;
    }

    public void setPage(int page) {
        this.page = page;
    }
//用于处理分页的方法
    public void list() {
        try {
            JSONObject root = new JSONObject();
            PageList<Admin> list = adminService.list(like, (page - 1) * rows,
                    rows);
            int count = list.getRows();
            root.put("page", page);
            root.put("total", count / rows + ((count % rows) > 0 ? 1 : 0));
            root.put("records", count);
            JSONArray rows = new JSONArray();
            for (Admin s : list) {
                rows.add(s, JsonUtil.config());
            }
            root.put("rows", rows);

    ServletActionContext.getResponse().setCharacterEncoding("utf-8");
            root.write(ServletActionContext.getResponse().getWriter());
        } catch (Throwable e) {
            e.printStackTrace();
        }

    }
    //添加 admin 信息
    public String add() throws ModelException {
```

```
        adminService.add(admin);
        return "success";
    }
    //修改 admin 信息
    public String modify() {
        try {
            adminService.modify(admin);
            return "success";
        } catch (ModelException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return "error";

    }

    //加载 admin 信息
   private int id;

    public void setId(int id) {
        this.id = id;
    }
    public void load() {
        try {
            JSONObject root = new JSONObject();
            Admin adm = adminService.load(id);
            root.element("admin", adm, JsonUtil.config());

    ServletActionContext.getResponse().setCharacterEncoding("utf-8");
            root.write(ServletActionContext.getResponse().getWriter());
        } catch (Throwable e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

(7) 创建 admin 的 action 映射文件。在项目中新建文件夹 action，在该文件夹下新建 AdminManager.xml 的配置文件，主要在配置 admin 的 action 类，代码如下。

```
<struts>
  <package name="AdminManager" extends="default" namespace="/admin">
     <action name="main" class="adminManager" method="main">
        <result name="success">/WEB-INF/jsp/Admin/main.jsp</result>
     </action>
     <action name="list" class="adminManager"  method="list"/>
      <action name="add" class="adminManager" method="add">
        <result name="success" type="redirect">main</result>
```

```
        </action>
        <action name="modify" class="adminManager" method="modify">
          <result name="success" type="redirect">main</result>
        </action>
       <action name="load" class="adminManager" method="load" />

   </package>
</struts>
```

以上文件配置好后，把该文件再在 struts.xml 中进行配置，以后这种各个模块的 action 配置文件，都要在 struts.xml 中进行配置。代码如下。

```
</package>
    <include file="action/Main.xml"/>
<package>
```

(8) 创建 admin 的 spring 映射文件。在项目中新建 bean 文件夹，在该文件夹中新建 Admin.xml，再改文件中配置需要用 Spring 进行管理的 dao 组件，service 组件和 action 组件。代码如下。

```
  <!-- DAO -->
  <bean id="adminDao" class="com.pmanager.dao.AdminDaoImpl">
   <property name="hibernateTemplate" ref="hibernateTemplate" />

  </bean>
  <!-- Services -->
  <bean id="adminService" class="com.pmanager.service.AdminServiceImpl">
    <property name="adminDao" ref="adminDao" />

  </bean>
  <!-- Actions -->
  <bean id="adminManager" class="com.pmanager.action.AdminManager"
scope="prototype">
    <property name="adminService" ref="adminService"></property>
  </bean>
</beans>
```

配置完后还要在 spring.xml 中进行配置，以后各个模块配置都要在 spring.xml 文件中进行配置。代码如下。

```
  <!-- 各个模块 Bean 声明的导入 -->
  <import resource="classpath:/beans/Admin.xml"/>
<import resource="classpath:/beans/Main.xml"/>
  <import resource="classpath:/beans/Emp.xml"/>
</bean>
```

(9) 前台页面的编写。在项目中的 WEB-INFO 文件下新建名为 jsp 的文件，在该文件中新建 Admin 文件，在该文件中新建 main.jsp，用于分页显示数据的代码、过滤查询和修改的代码如下。

```
<body>
    <div id="tabs">
      <ul>
        <li><a href="#tab-1">管理员列表</a></li>
        <li><a href="#tab-2">添加管理员</a></li>
        <li><a href="#tab-3">修改管理员</a></li>
      </ul>
      <div id="tab-1">
   <table id="grid" class="table" width="100%" cellspacing=0 cellpadding=0>
              </table>
              <div id="page"></div>
        </div>
        <div id="tab-2">
          <form action="add" method="post" id="add_form">
          <s:hidden name="admin.id"/>
          <table class="table" width="100%" cellspacing=0 cellpadding=0>
              <tr class="tr">
                <th class="th" colspan="4" align="left">  管理员信息</th>
              </tr>
              <tr class="tr">
                <td class="td">用户名:</td><td class="td"><s:textfield
name="admin.adminname" /></td>
                <td class="td">密    码:</td><td class="td"><s:textfield
name="admin.password" /></td>

              </tr>
          <tr class="tr">
            <td class="td" style="height:90px;" colspan="4"
align="center"><button id="add_save">保存</button><button id="add_reset">重置
</button></td>
          </tr>
          </table>
          </form>
        </div>
        <div id="tab-3">
          <form action="modify" method="post" id="mod_form">
           <s:hidden name="admin.id" id="adminid"/>
          <table class="table" width="100%" cellspacing=0 cellpadding=0>
              <tr class="tr">
                <th class="th" colspan="4" align="left">  管理员信息</th>
              </tr>
              <tr class="tr">
                <td class="td">用户名:</td><td class="td"><s:textfield
id="adminadminname" name="admin.adminname" /></td>
                <td class="td">密    码:</td><td class="td"><s:textfield
id="adminpassword" name="admin.password" /></td>
              </tr>
          <tr class="tr">
```

```
          <td class="td" style="height:90px;" colspan="4"
align="center"><button id="mod_save">保存</button><button id="mod_reset">重置
</button></td>
        </tr>
         </table>
         </form>
       </div>
   </div>
   <script type="text/javascript">
   $(function(){
      $('#tabs').height($(window).height()-10);
      $('#tabs').tabs();
      $('#tabs').tabs('disable',2)
      $('button').button();

      init_table();
      init_add();
      init_mod();

   });
   function init_add(){
     $('#add_save').click(function(){
        $('#add form').submit();
           return false;
      });
     $('#add_reset').click(function(){
           $('#tabs').tabs( "select" , 0);
            return false;
      });
   }
   function init_mod(){
     $('#mod_save').click(function(){
         $('#mod_form').submit();
         return false;
     });
     $('#mod_reset').click(function(){
         $('#tabs').tabs("select",0);
         $('#tabs').tabs('disable',2);
         return false;
     });
   }
   function init table(){
    $("#grid").jqGrid({
      url:'list',
      datatype : "json",
        colModel:[
          {name:'like.id',jsonmap:"id",label:'编号',width:40,align:"left"},
   {name:'like.adminname',jsonmap:"adminname",label:'用户名
',width:60,align:"left"},
```

```
        {name:'like.password',jsonmap:"password",label:'密码
',width:175,align:"left"},
       ],
       scroll: true,
       pager: '#page',
       viewrecords: true,
     caption: "管理员查询结果",
     width: $(window).width()-40,
     height: $(window).height()-170,
     forceFit : true,
     multiselect:true,
     jsonReader:{
         repeatitems : false
      },
      ondblClickRow:function(id,row,col,e){

        load(id,function(){
            $('#tabs').tabs('enable',2);
            $('#tabs').tabs("select",2);
        });
      }
   });
   $('#grid').navGrid('#page',{
       add:false,
       edit:false,
       search:false,
       delfunc:function(ids){
           remove(ids);
       }
   });
   $('#grid').filterToolbar();
  }
  function remove(id){
     alert(id);
  }
  function load(id,callback){
   $.getJSON('load',{id:id},function(data,status,xhr){
       var admin = data.admin;
       for(var v in admin){
       alert(admin[v]);
         try{$('#admin'+v).val(admin[v]);}catch(e){alert(e);}
       }
      callback();
    });
  }
  </script>
 </body>
</html>
```

至此，管理员管理模块就完成了，下面来看看效果。单击“管理员信息管理”按钮，出现如图 15-5 所示的页面。



图 15-5　管理员信息管理

单击“添加管理员”按钮，系统自动进入如图 15-6 所示的界面。通过该界面可以添加管理员信息，包括管理员的用户名和密码，如图 15-6 所示。

在本系统中，按如图 15-5 所示，双击要修改的管理员信息所在的行，可以进入修改管理员界面。通过该界面可以对所选中的管理员的信息进行修改，如图 15-7 所示。



图 15-6　添加管理员　　　　**图 15-7　修改管理员**

在管理员列表中，通过☑勾选要删除的管理员信息列，然后单击按钮，系统会提示要删除的管理员信息列，如图 15-8 所示。



图 15-8　删除管理员信息

15.4.2 员工管理模块——jQuery框架的使用

前一小节，讲解了模块后台处理程序的实现过程。在这一节里，将介绍前台友好界面的具体实现过程，在此将用到一个优秀的 JavaScript 框架——jQuery。

首先，先浏览下员工管理模块的前台展示效果，员工管理模块主要包括员工列表、添加员工、修改员工和删除员工四部分。单击“员工信息管理”和“员工列表”按钮可以看到员工信息的表单，如图 15-9 所示。



图 15-9 员工信息管理

如图 15-9 所示，按钮的效果及左侧栏中的菜单栏中的下拉框的效果都是通过 jQuery 来实现的，下面来简单地介绍一下其实现过程。

(1) 在 jsp 页面中导入 jQuery 包，这里要用到四个包：jquery-1.4.2.min.js、jquery-ui-1.8.5.min.js、grid.locale-cn.js、jquery.jqgrid.min.js。在页面中使用这些包的方法代码如下。

```
<script type="text/javascript"
src="../scripts/jquery-1.4.2.min.js"></script>
  <script type="text/javascript"
src="../scripts/jquery-ui-1.8.5.min.js"></script>
  <script type="text/javascript" src="../scripts/i18n/grid.locale-cn.js"
charset="utf-16"></script>
  <script type="text/javascript" src="../scripts/jquery.jqgrid.min.js"
charset="utf-8"></script>
```

(2) 页面展示效果的实现。页面的部分代码如下。

```
<body>
    <div id="tabs">
      <ul>
        <li><a href="#tab-1">管理员列表</a></li>
        <li><a href="#tab-2">添加管理员</a></li>
        <li><a href="#tab-3">修改管理员</a></li>
      </ul>
      <div id="tab-1">
```

```
            <table id="grid" class="table" width="100%" cellspacing=0
cellpadding=0>
            </table>
            <div id="page"></div>
      </div>
        <div id="tab-2">
          <form action="add" method="post" id="add_form">
          <s:hidden name="admin.id"/>
          <table class="table" width="100%" cellspacing=0 cellpadding=0>
              <tr class="tr">
                <th class="th" colspan="4" align="left"> 管理员信息</th>
              </tr>
              <tr class="tr">
                <td class="td">用户名:</td><td class="td"><s:textfield
name="admin.adminname" /></td>
                <td class="td">密    码:</td><td class="td"><s:textfield
name="admin.password" /></td>

              </tr>
         <tr class="tr">
           <td class="td" style="height:90px;" colspan="4"
align="center"><button id="add_save">保存</button><button id="add_reset">重置
</button></td>
         </tr>
          </table>
          </form>
        </div>
        <div id="tab-3">
          <form action="modify" method="post" id="mod_form">
           <s:hidden name="admin.id" id="adminid"/>
          <table class="table" width="100%" cellspacing=0 cellpadding=0>
              <tr class="tr">
                <th class="th" colspan="4" align="left"> 管理员信息</th>
              </tr>
              <tr class="tr">
                <td class="td">用户名:</td><td class="td"><s:textfield
id="adminadminname" name="admin.adminname" /></td>
                <td class="td">密    码:</td><td class="td"><s:textfield
id="adminpassword" name="admin.password" /></td>
              </tr>
         <tr class="tr">
           <td class="td" style="height:90px;" colspan="4"
align="center"><button id="mod_save">保  存</button><button id="mod_reset">重
置</button></td>
         </tr>
          </table>
          </form>
        </div>
    </div>
```

根据以上代码，给 id 为 tab-1 的 div 中的 table 定义 id 为 grid，通过后台程序处理得到的数据以 json 数据形式向前台输入，然后将对应的 init_table 函数在初始化函数中初始化。其部分代码如下。

```
function init_table(){
    $("#grid").jqGrid({
      url:'list',
      datatype : "json",
        colModel:[
          {name:'like.id',jsonmap:"id",label:'编号',width:40,align:"left"},
         {name:'like.adminname',jsonmap:"adminname",label:'用户名
',width:60,align:"left"},
          {name:'like.password',jsonmap:"password",label:'密码
',width:175,align:"left"},

        ],
        scroll: true,
        pager: '#page',
        viewrecords: true,
      caption: "管理员查询结果",
      width: $(window).width()-40,
      height: $(window).height()-170,
      forceFit : true,
      multiselect:true,
      jsonReader:{
          repeatitems : false
       },
       ondblClickRow:function(id,row,col,e){

         load(id,function(){
             $('#tabs').tabs('enable',2);
             $('#tabs').tabs("select",2);
         });
       }
    });
    $('#grid').navGrid('#page',{
        add:false,
        edit:false,
        search:false,
        delfunc:function(ids){
            remove(ids);
        }
    });
    $('#grid').filterToolbar();
   }
```

项目运行时，前台向后台传递 action 请求，Struts 2 根据请求找到与请求相应的 action，通过 action 与业务逻辑的交互，再把请求响应传到页面，因此我们就看到了以上页面。其中的 action 也是继承了 com.opensymphony.xwork2.ActionSupport 类，代码如下。

```
public class EmpManager extends ActionSupport {
  private EmpService empService = null;
  private Employee employee = null;
  private String birthday;
   public Employee getEmployee() {
       return employee;
   }
   public void setEmployee(Employee employee) {
       this.employee = employee;
   }
   public void setEmpService(EmpService empService) {
       this.empService = empService;
   }
   public void setBirthday(String birthday) {
       this.birthday = birthday;
   }
  /**
   *  显示员工主界面
   * @return  success
   */
   public String main(){
       return "success";
   }
   //过滤查询功能
   private int rows = 10;
   private int page = 1;
   private Employee like = new Employee();
   private Short id;
   public void setId(Short id) {
       this.id = id;
   }
   public Employee getLike() {
       return like;
   }
   public void setLike(Employee like) {
       this.like = like;
   }
   public void setRows(int rows) {
       this.rows = rows;
   }
   public void setPage(int page) {
       this.page = page;
   }
   public void list() {
       try {
           JSONObject root = new JSONObject();
           PageList<Employee> list = empService.list(like, (page - 1) * rows,
                   rows);
```

```
            int count = list.getRows();
            root.put("page", page);
            root.put("total", count / rows + ((count % rows) > 0 ? 1 : 0));
            root.put("records", count);
            JSONArray rows = new JSONArray();
            for (Employee s : list) {
                rows.add(s, JsonUtil.config());
            }
            root.put("rows", rows);

    ServletActionContext.getResponse().setCharacterEncoding("utf-8");
            root.write(ServletActionContext.getResponse().getWriter());
        } catch (Throwable e) {
            e.printStackTrace();
        }

    }
    public String add() throws ModelException {

        empService.add(employee);
        return "success";
    }
    public String modify() {
        try {

            empService.modify(employee);
            return "success";
        } catch (ModelException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return "error";
    }
    public void load() {
        try {
            JSONObject root = new JSONObject();
            Employee emp = empService.load(id);
            root.element("employee", emp, JsonUtil.config());

    ServletActionContext.getResponse().setCharacterEncoding("utf-8");
            root.write(ServletActionContext.getResponse().getWriter());
        } catch (Throwable e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

单击“添加员工”按钮，系统自动进入如图 15-10 所示的界面。通过该界面可以添加员工信息，包括员工的姓名、密码、性别、生日、地址、电话和简介，如图 15-10 所示。

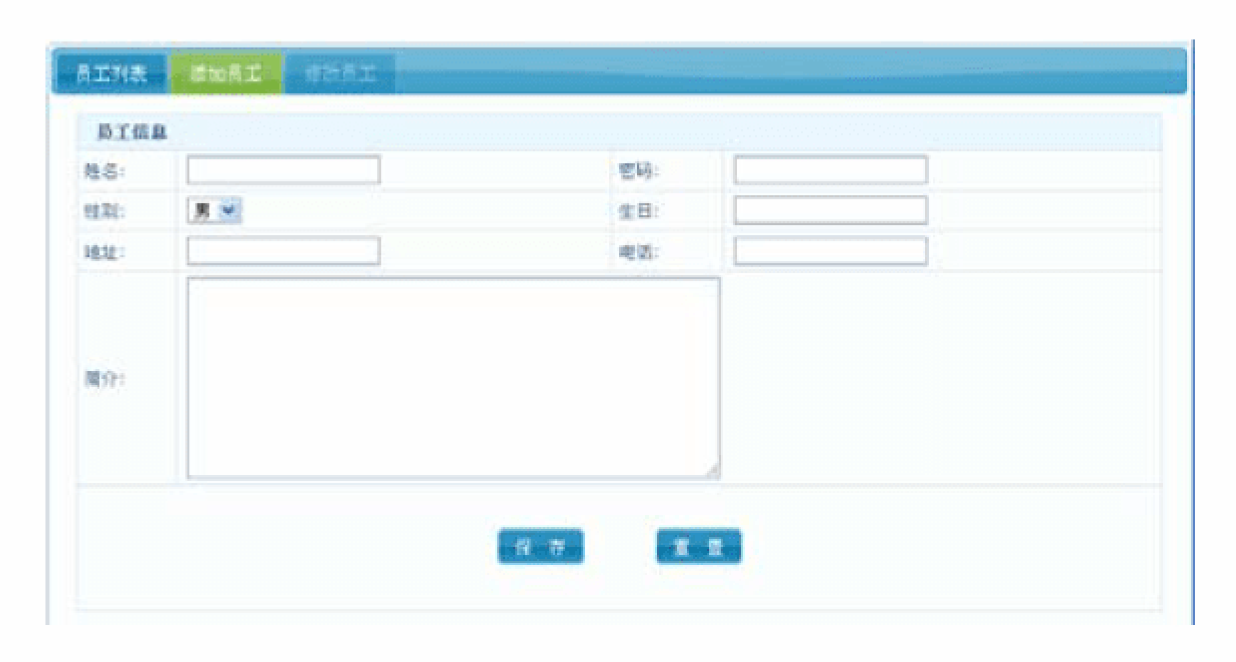


图 15-10　添加员工

当填写完要输入的信息后，单击“提交”按钮，根据前台 JavaScript 的 init_add 函数为按钮附加一个 jQuery 事件，然后调用相应的 action 中的 add 方法进行处理添加信息，代码如下。

```
function init_add(){
     $('#add_save').click(function(){
        $('#add_form').submit();
            return false;
       });
     $('#add_reset').click(function(){
           $('#tabs').tabs( "select" , 0);
            return false;
       });
   }
```

在本系统中，按如图 15-9 所示，双击要修改的员工信息所在的行，可以进入修改员工界面。通过该界面可以对所选中的员工的信息进行修改，包括员工的姓名、密码、性别、生日、地址、电话和简介，如图 15-11 所示。



图 15-11　修改员工

填写完要修改的信息后，根据前台 main 页面的 JavaScript 的 init_mod()方法，为“提交”按钮附加事件，然后跳转到后 action 进行处理。代码如下。

```
function init_mod(){
     $('#mod_save').click(function(){
         $('#mod_form').submit();
         return false;
     });
```

```
    $('#mod_reset').click(function(){
        $('#tabs').tabs("select",0);
        $('#tabs').tabs('disable',2);
        return false;
    });
}
```

在员工列表中，通过勾选要删除的员工信息列，然后单击按钮，系统会提示要删除的员工信息列，如图 15-12 所示。

图 15-12　删除员工信息

通过以上过程我们了解了这个项目中的整体模块的开发步骤，也看到了使用 jQuery 框架所展示的前台的效果。以下章节可以在前两小节介绍的基础上进行逐一开发，下面在这里简单的介绍下剩余模块的功能和简单实现方式。

15.4.3　应聘管理模块

应聘管理模块主要包括应聘者列表、添加应聘者、修改应聘者和删除应聘四部分。单击“应聘信息管理”和“应聘者列表”按钮可以看到应聘者信息的表单，如图 15-13 所示。

图 15-13　应聘者信息管理

以上列表的显示，前台向后台传递了 action 请求，Struts 2 根据请求找到与请求相应的 action，通过 action 与业务逻辑的交互，再把请求响应传到页面，因此就看到了以上页面。其中的 action 也是继承了 com.opensymphony.xwork2.ActionSupport 类，代码如下。

```
public class RecruitmentManager extends ActionSupport {
   private RecruitmentService recruitmentService = null;
   private Recruitment recruitment = null;
   public Recruitment getRecruitment() {
         return recruitment;
    }
    public void setRecruitment(Recruitment recruitment) {
         this.recruitment = recruitment;
    }
    public void setRecruitmentService(RecruitmentService recruitmentService)
{
         this.recruitmentService = recruitmentService;
    }
/**
    *  显示员工主界面
    * @return  success
    */
    public String main(){
         return "success";
    }
    //过滤查询功能
    private int rows = 10;
    private int page = 1;
    private Recruitment like = new Recruitment();
    public Recruitment getLike() {
         return like;
    }
    public void setLike(Recruitment like) {
         this.like = like;
    }
    public void setRows(int rows) {
         this.rows = rows;
    }
    public void setPage(int page) {
         this.page = page;
    }
    public void list() {
         try {
             JSONObject root = new JSONObject();
             PageList<Recruitment> list = recruitmentService.list(like, (page
- 1) * rows,
                     rows);
             int count = list.getRows();
             root.put("page", page);
             root.put("total", count / rows + ((count % rows) > 0 ? 1 : 0));
             root.put("records", count);
             JSONArray rows = new JSONArray();
             for (Recruitment s : list) {
```

```
                rows.add(s, JsonUtil.config());
            }
            root.put("rows", rows);

    ServletActionContext.getResponse().setCharacterEncoding("utf-8");
            root.write(ServletActionContext.getResponse().getWriter());
        } catch (Throwable e) {
            e.printStackTrace();
        }
    }

    public String add() throws ModelException {

        recruitmentService.add(recruitment);
        return "success";
    }
    public String modify() {
        try {
            recruitmentService.modify(recruitment);
            return "success";
        } catch (ModelException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return "error";
    }

    //加载
   private int id;

    public void setId(int id) {
        this.id = id;
    }
    public void load() {
        try {
            JSONObject root = new JSONObject();
            Recruitment rec = recruitmentService.load(id);
            root.element("recruitment", rec, JsonUtil.config());

    ServletActionContext.getResponse().setCharacterEncoding("utf-8");
            root.write(ServletActionContext.getResponse().getWriter());
        } catch (Throwable e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

单击“添加应聘者”按钮，系统自动进入如图 15-14 所示的界面。通过该界面可以添加应聘者信息，包括应聘者的姓名、性别、年龄、职位、专业、工作经验、学历、学校、电话、邮

箱和简介，如图 15-14 所示。

图 15-14　添加应聘者

当填写完要输入的信息后，单击“提交”按钮，根据前台 main 页面中的 JavaScript 的 add 函数为按钮附加一个 jQuery 事件，然后调用相应的 action 中的 add 方法进行处理添加信息，代码如下。

```
function init_add(){
     $('#add_save').click(function(){
        $('#add_form').submit();
            return false;
      });
     $('#add_reset').click(function(){
           $('#tabs').tabs( "select" , 0);
            return false;
      });
   }
```

在本系统中，按如图 15-13 所示，双击要修改的应聘者信息所在的行，可以进入修改应聘者界面。通过该界面可以对所选中的应聘者的信息进行修改，包括应聘者的姓名、性别、年龄、职位、专业、工作经验、学历、学校、电话、邮箱和简介，如图 15-15 所示。

图 15-15　修改应聘者

在应聘者列表中，通过☑勾选要删除的应聘者信息列，然后单击按钮，系统会提示要删除的应聘者信息列，如图 15-16 所示。

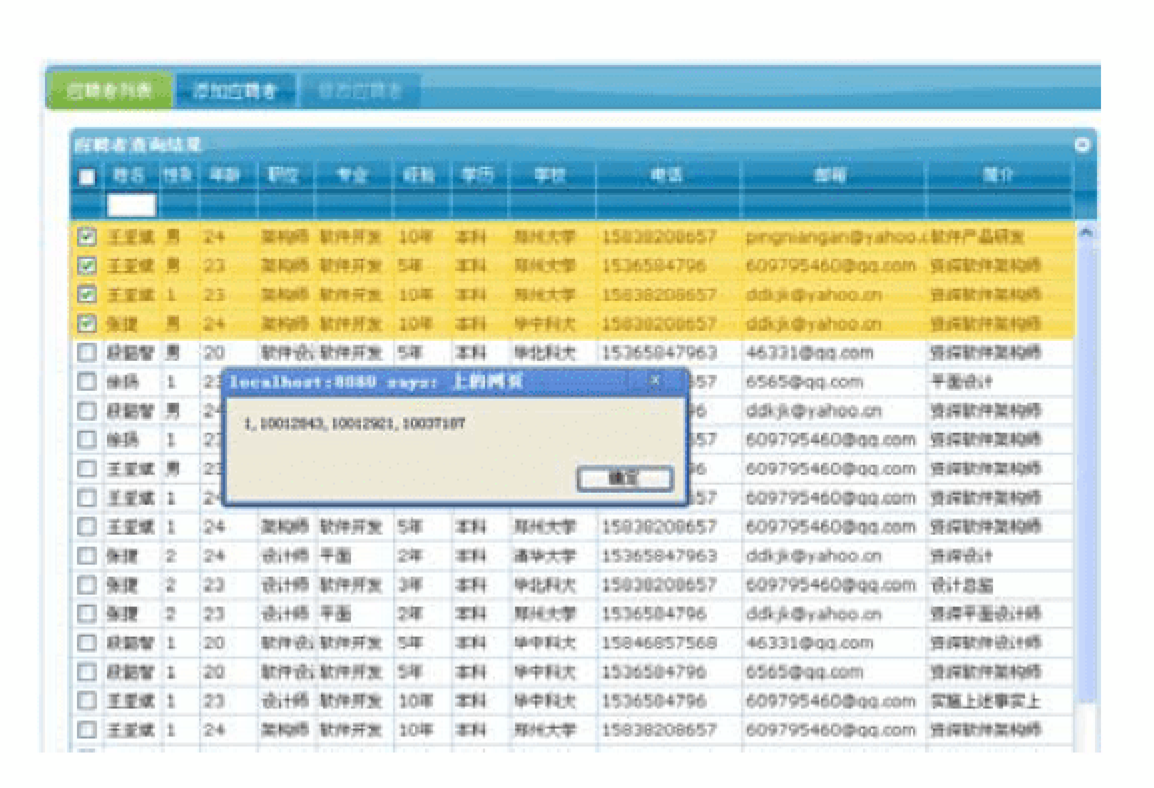


图 15-16　删除应聘者

15.4.4　奖惩管理模块

奖惩管理模块主要包括奖惩列表、添加奖惩、修改奖惩和删除奖惩四部分。单击“奖惩信息管理”和“奖惩列表”按钮可以看到奖惩信息的表单，如图 15-17 所示。



图 15-17　奖惩信息管理

以上列表的显示，前台向后台传递了 action 请求，Struts 2 根据请求找到与请求相应的 action，通过 action 与业务逻辑的交互，再把请求响应传到页面，我们就看到了以上页面。其中的 action 也是继承了 com.opensymphony.xwork2.ActionSupport 类，代码如下。

```
public class IncentivesManager extends ActionSupport {
   private IncentivesService incentivesService = null;
   private Incentives incentives = null;
    public Incentives getIncentives() {
        return incentives;
    }
    public void setIncentives(Incentives incentives) {
        this.incentives = incentives;
    }
    public void setIncentivesService(IncentivesService incentivesService) {
```

```
        this.incentivesService = incentivesService;
    }
/**
    *  显示员工主界面
    * @return  success
    */
    public String main(){
        return "success";
    }
    //过滤查询功能
    private int rows = 10;
    private int page = 1;
    private Incentives like = new Incentives();



    public Incentives getLike() {
        return like;
    }
    public void setLike(Incentives like) {
        this.like = like;
    }
    public void setRows(int rows) {
        this.rows = rows;
    }
    public void setPage(int page) {
        this.page = page;
    }
    public void list() {
        try {
            JSONObject root = new JSONObject();
            PageList<Incentives> list = incentivesService.list(like, (page -
1) * rows,
                    rows);
            int count = list.getRows();
            root.put("page", page);
            root.put("total", count / rows + ((count % rows) > 0 ? 1 : 0));
            root.put("records", count);
            JSONArray rows = new JSONArray();
            for (Incentives s : list) {
                rows.add(s, JsonUtil.config());
            }
            root.put("rows", rows);
    ServletActionContext.getResponse().setCharacterEncoding("utf-8");
    root.write(ServletActionContext.getResponse().getWriter());
        } catch (Throwable e) {
            e.printStackTrace();
        }
    }
```

```
    public String add() throws ModelException {

        incentivesService.add(incentives);
        return "success";
    }
    public String modify() {
        try {
            incentivesService.modify(incentives);
            return "success";
        } catch (ModelException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return "error";
    }
    //加载
   private int id;
    public void setId(int id) {
        this.id = id;
    }
    public void load() {
        try {
            JSONObject root = new JSONObject();
            Incentives ince = incentivesService.load(id);
            root.element("incentives", ince, JsonUtil.config());
    ServletActionContext.getResponse().setCharacterEncoding("utf-8");
    root.write(ServletActionContext.getResponse().getWriter());
        } catch (Throwable e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

单击“添加奖惩”按钮，系统自动进入如图 15-18 所示的界面。通过该界面可以添加奖惩信息，包括奖惩的奖惩名称、奖惩原因和奖惩说明，如图 15-18 所示。

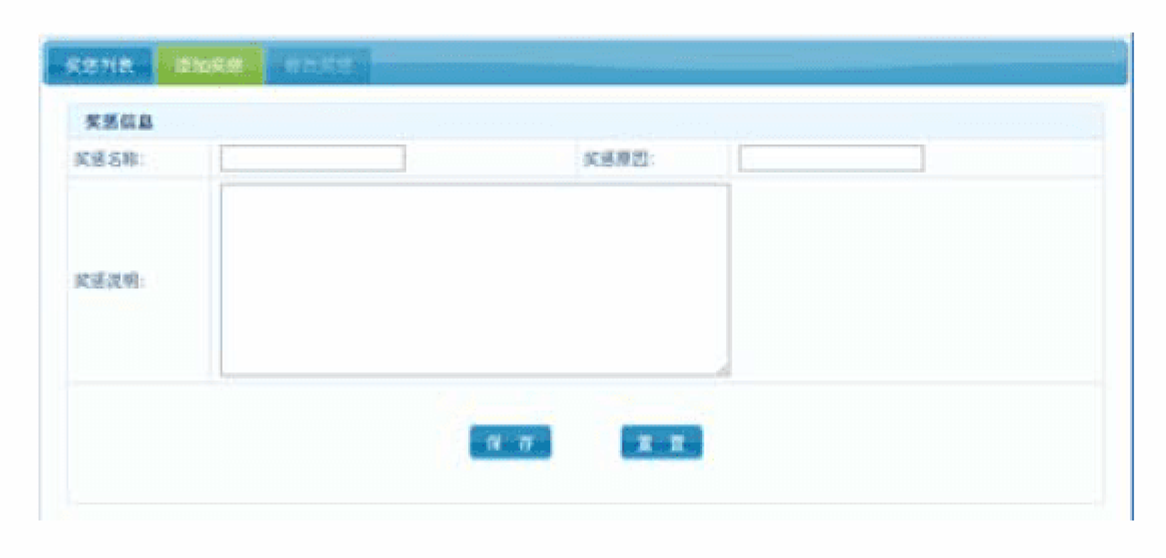


图 15-18　添加奖惩

在本系统中，按如图 15-17 所示，双击要修改的奖惩信息所在的行，可以进入修改奖惩界面。通过该界面可以对所选中的奖惩的信息进行修改，包括奖惩的奖惩名称、奖惩原因和奖惩说明，如图 15-19 所示。



图 15-19　修改奖惩

在奖惩列表中，通过勾选要删除的奖惩信息列，然后单击按钮，系统会提示要删除的奖惩信息列，如图 15-20 所示。

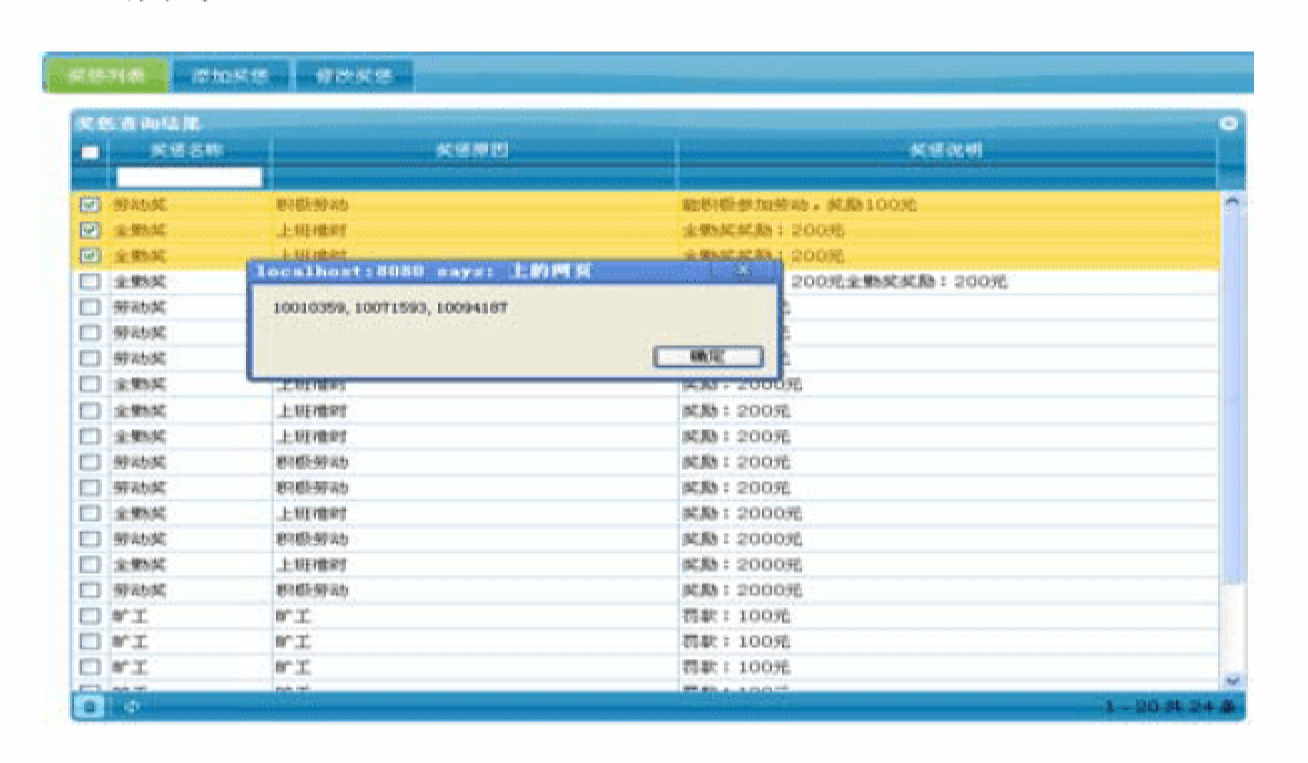


图 15-20　删除奖惩

15.4.5　培训管理模块

培训理模块主要包括培训列表、添加培训、修改培训和删除培训四部分。单击“培训信息管理”和“培训列表”按钮可以看到培训信息的表单，如图 15-21 所示。



图 15-21　培训信息管理

以上列表的显示，前台向后台传递了 action 请求，Struts 2 根据请求找到与请求相应的

action，通过 action 与业务逻辑的交互，再把请求响应传到页面，我们就看到了以上页面。其中的 action 也是继承了 com.opensymphony.xwork2.ActionSupport 类，代码如下。

```
public class TrainManager extends ActionSupport {

   private TrainService trainService = null;
    private Train train = null;
    private String starttime;
    private String stoptime;
    public Train getTrain() {
        return train;
    }   public void setTrain(Train train) {
        this.train = train;
    }   public void setTrainService(TrainService trainService) {
        this.trainService = trainService;
    }
    public String getStarttime() {
        return starttime;
    }
    public void setStarttime(String starttime) {
        this.starttime = starttime;
    }
    public String getStoptime() {
        return stoptime;
    }   public void setStoptime(String stoptime) {
        this.stoptime = stoptime;
    }/**
   *  显示员工主界面
   * @return  success
   */
    public String main(){
        return "success";
    }
    //过滤查询功能
    private int rows = 10;
    private int page = 1;
    private Train like = new Train();
    public Train getLike() {
        return like;
    }
    public void setLike(Train like) {
        this.like = like;
    }
    public void setRows(int rows) {
        this.rows = rows;
    }
    public void setPage(int page) {
        this.page = page;
    }
```

```
    public void list() {
        try {
            JSONObject root = new JSONObject();
            PageList<Train> list = trainService.list(like, (page - 1) * rows,
                    rows);
            int count = list.getRows();
            root.put("page", page);
            root.put("total", count / rows + ((count % rows) > 0 ? 1 : 0));
            root.put("records", count);
            JSONArray rows = new JSONArray();
            for (Train s : list) {
                rows.add(s, JsonUtil.config());
            }
            root.put("rows", rows);

    ServletActionContext.getResponse().setCharacterEncoding("utf-8");
            root.write(ServletActionContext.getResponse().getWriter());
        } catch (Throwable e) {
            e.printStackTrace();
        }
    }
    public String add() throws ModelException {


        trainService.add(train);
        return "success";
    }

    public String modify() {
        try {


            trainService.modify(train);
            return "success";
        } catch (ModelException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return "error";
    }

    //加载
   private int id;
    public void setId(int id) {
        this.id = id;
    }
    public void load() {
        try {
            JSONObject root = new JSONObject();
```

```
            Train tra = trainService.load(id);
            root.element("train", tra, JsonUtil.config());

    ServletActionContext.getResponse().setCharacterEncoding("utf-8");
            root.write(ServletActionContext.getResponse().getWriter());
        } catch (Throwable e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

单击“添加培训”按钮，系统自动进入如图 15-22 所示的界面。通过该界面可以添加培训信息，包括培训名称、培训目的、开始时间、结束时间、讲师、培训人员和培训内容，在该页面填写完成后，单击“提交”按钮，调用以上 action 的 add 方法，进行添加。如图 15-22 所示。

图 15-22 添加培训

在本系统中，按如图 15-21 所示，双击要修改的培训信息所在的行，可以进入修改培训界面。通过该界面可以对所选中的培训的信息进行修改，包括培训的培训名称、培训目的、开始时间、结束时间、讲师、培训人员和培训内容，如图 15-23 所示。

图 15-23 修改培训

在培训列表中，通过勾选要删除的培训信息列，然后单击按钮，系统会提示要删除的培训信息列，如图 15-24 所示。

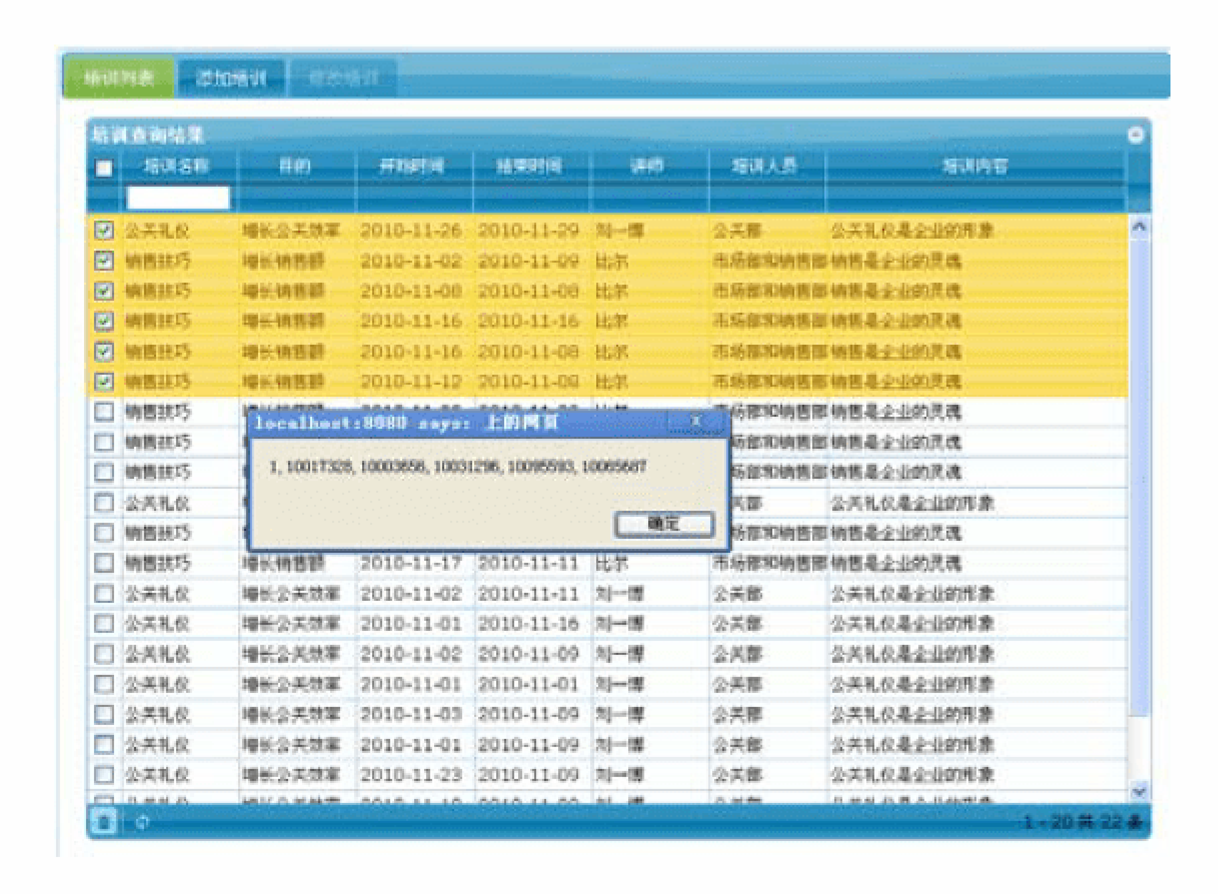


图 15-24　删除培训信息

15.4.6　薪资管理模块

薪资管理模块主要包括薪资列表、添加薪资、修改薪资和删除薪资四部分。单击“薪资信息管理”和“薪资列表”按钮可以看到培训信息的表单，包括员工姓名、基本工资、食补、房补、全勤奖、罚款、发放时间和总计等信息，如图 15-25 所示。

图 15-25　薪资信息管理

以上列表的显示，前台向后台传递了 action 请求，Struts 2 根据请求找到与请求相应的 action，通过 action 与业务逻辑的交互，再把请求响应传到页面，我们就看到了以上页面。其中的 action 也是继承了 com.opensymphony.xwork2.ActionSupport 类，代码如下。

```
@SuppressWarnings("serial")
public class PayManager extends ActionSupport {
    private PayService payService = null;
     private Pay pay = null;
     private String starttime;
     private String stoptime;
```

```
    public Pay getPay() {
        return pay;
    }
    public void setPay(Pay pay) {
        this.pay = pay;
    }
    public void setPayService(PayService payService) {
        this.payService = payService;
    }
    public String getStarttime() {
        return starttime;
    }
    public void setStarttime(String starttime) {
        this.starttime = starttime;
    }
    public String getStoptime() {
        return stoptime;
    }
    public void setStoptime(String stoptime) {
        this.stoptime = stoptime;
    }
    /**
   *  显示员工主界面
   * @return  success
   */
    public String main(){
        return "success";
    }
    //过滤查询功能
    private int rows = 10;
    private int page = 1;
    private Pay like = new Pay();
    public Pay getLike() {
        return like;
    }
    public void setLike(Pay like) {
        this.like = like;
    }
    public void setRows(int rows) {
        this.rows = rows;
    }   public void setPage(int page) {
        this.page = page;
    }
    public void list() {
        try {
            JSONObject root = new JSONObject();
            PageList<Pay> list = payService.list(like, (page - 1) * rows,
                    rows);
            int count = list.getRows();
```

```
            root.put("page", page);
            root.put("total", count / rows + ((count % rows) > 0 ? 1 : 0));
            root.put("records", count);
            JSONArray rows = new JSONArray();
            for (Pay s : list) {
                rows.add(s, JsonUtil.config());
            }
            root.put("rows", rows);

    ServletActionContext.getResponse().setCharacterEncoding("utf-8");
            root.write(ServletActionContext.getResponse().getWriter());
        } catch (Throwable e) {
            e.printStackTrace();
        }   }
    public String add() throws ModelException {
        payService.add(pay);
        return "success";
    }
    public String modify() {
        try {
            payService.modify(pay);
            return "success";
        } catch (ModelException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return "error";
    }
    //加载
   private int id;
    public void setId(int id) {
        this.id = id;
    }
    public void load() {
        try {
            JSONObject root = new JSONObject();
            Pay pa = payService.load(id);
            root.element("pay", pa, JsonUtil.config());

    ServletActionContext.getResponse().setCharacterEncoding("utf-8");
            root.write(ServletActionContext.getResponse().getWriter());
        } catch (Throwable e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

单击“添加薪资”按钮，系统自动进入如图 15-26 所示的界面。通过该界面可以添加薪资信息，包括薪资的员工姓名、基本工资、食补、房补、全勤奖、罚款、发放时间和总计，如

图 15-26 所示。

图 15-26　添加薪资

在本系统中，按如图 15-25 所示，双击要修改的薪资信息所在的行，可以进入修改薪资界面。通过该界面可以对所选中的薪资的信息进行修改，包括薪资的员工姓名、基本工资、食补、房补、全勤奖、罚款、发放时间和总计，如图 15-27 所示。



图 15-27　修改薪资

在薪资列表中，通过勾选要删除的薪资信息列，然后单击按钮，系统会提示要删除的薪资信息列，如图 15-28 所示。

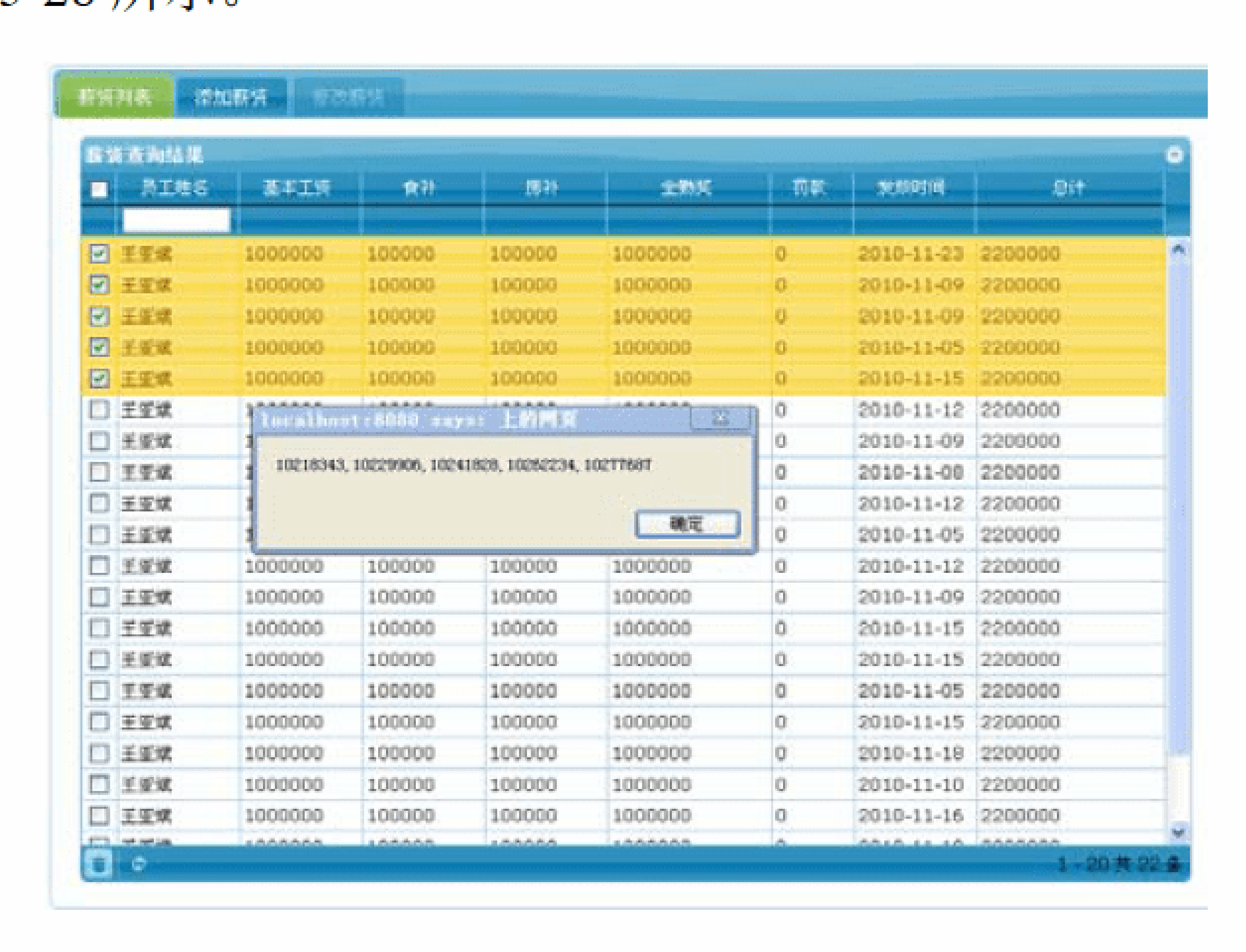


图 15-28　删除薪资信息

15.5 总　　结

人力资源管理系统是企业协同管理平台的重要组成部分，是提高人力资源管理水平而设计开发的企业信息管理系统。系统开发采取 Java 平台，采用 Struts 2、Spring、Hibernate(SSH)的框架，本系统是在 Struts 2 的环境下着重对系统进行分析和讲解的。具体分析设计了员工管理、招聘管理、培训管理、奖惩管理、薪资管理和管理员管理六大模块，系统基本能满足企业的要求。但是在开发过程中，还是显露出很多的问题，如在开发过程中，对 Struts 2 监听器的使用是否熟练和系统代码的实际问题，将有待深入地学习 Struts 2 会得以解决。

附录　参 考 答 案

第 1 章

一、填空题

(1)　Struts 2-core-2.1.8.1.jar
(2)　struts.xml
(3)　用来配置 Action 类
(4)　ActionSupport

二、选择题

(1)　A
(2)　D
(3)　B

第 2 章

一、填空题

(1)　org.apache.struts2.dispatcher.FilterDispatcher
(2)　<include file="struts-default.xml"></include>
(3)　namespace
(4)　Execute()
(5)　dispatcher
(6)　<exception-mapping…>

二、选择题

(1)　B
(2)　D
(3)　A
(4)　C

第 3 章

一、填空题

(1) 使用 JDK1.5 的注释来注册类型转换器
(2) KeyProperty_students=name
(3) conversionError
(4) xwork.default.invalid.fieldvalue

二、选择题

(1) B
(2) A
(3) C

第 4 章

一、填空题

(1) 本地化
(2) 国际化
(3) 占位符
(4) language
(5) RegisterAction
(6) ActionContext.getContext() .setLocale(Locale locale)

二、选择题

(1) A
(2) D
(3) D

第 5 章

一、填空题

(1) 拦截器名
(2) <interceptor-ref name="logger"/><interceptor-ref name="security"/>
(3) intercept
(4) doIntercept

(5) priority

二、选择题

(1) D
(2) A
(3) A
(4) B

第 6 章

一、填空题

(1) 在 validate()方法中进行校验
(2) 13
(3) Validator
(4) Address-cardid-validation.xml

二、选择题

(1) A
(2) D
(3) D
(4) C

第 7 章

一、填空题

(1) 根对象(Root Object)
(2) 数值常量
(3) 用于判断一个值是否在集合中
(4) :[表达式……]

二、选择题

(1) A
(2) D
(3) A
(4) C

第 8 章

一、填空题

(1) 控制标签
(2) java.util.Comparator
(3) ValueStack 栈顶
(4) scope
(5) xhtml

二、选择题

(1) B
(2) D
(3) A
(4) B
(5) C
(6) A
(7) C
(8) A
(9) A
(10) A
(11) C

第 9 章

一、填空题

(1) commons-fileupload
(2) FileUploadInterceptor
(3) image/gif,image/jpeg,image/png
(4) java.util.List
(5) inputName

二、选择题

(1) B
(2) A
(3) A
(4) D

第 10 章

一、填空题

(1) Token
(2) TokenSessionStoreInterceptor
(3) token
(4) tokenSession
(5) delaySleepInterval

二、选择题

(1) A
(2) C
(3) B

第 11 章

一、填空题

(1) 环绕通知
(2) HQL
(3) Cascade
(4) Hibernate
(5) Spring

二、选择题

(1) A
(2) C
(3) B
(4) C

第 12 章

一、填空题

(1) chart
(2) DefaultCategoryDataset
(3) TimeSeriesCollection

二、选择题

(1) C
(2) A
(3) C

第 13 章

一、填空题

(1) uk.ltd.getahead.dwr.DWRServlet
(2) 值的有序列表
(3) 字符串
(4) 将指定的事件处理函数绑定到事件上，也可以绑定到某个函数上，在被绑定的函数执行后，指定的函数将会被触发执行。

二、选择题

(1) C
(2) A
(3) B
(4) C
(5) B
(6) C
(7) B